Week1

2017年1月23日 15:57

Quick Review of Logrithon & Expotentials.

FACts: For real number n & b.

$\log_b n = k$ iff $b^k = n$.

— b is referred to as the base of Logrithm & expotentials.

→ In CS, $b=2$ so $\log n$ is $\log_2 n$.

Properties

SEC 1.3.2

1. $\log_b n$ is the inverse function of b^n and vice versa.

Ex: $2^{\log_2 n} = n$ $\log_2 2^n = n$.

2. Changing bases: $\log_b n = \frac{\log_c n}{\log_c b} = \frac{1}{\log_c b} \cdot \log_c n$

Ex $\log_2 n = \frac{1}{\log_3 2} \cdot \log_3 n$

$= 1.555 \log_3 n$

3. pulling exponents down.

$\log_b(n^c) = c \cdot \log_b n$. Ex: $\log_2 n^3 = 3\log_2 n$.

4. log products

$\log_b n_1 \cdot n_2 = \log_b n_1 + \log_b n_2$. Ex. $\log(2n\log n) = \log 2 + \log n + \log\log n$.

2

2017年1月23日 15:57

等比数列求和公式: Sum of the Geometric Progression

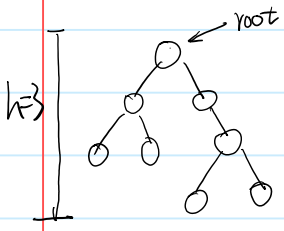
$$S_n = \begin{cases} na_1, (q = 1), \\ \frac{a_1 - a_1 q^n}{1 - q}, (q \neq 1). \end{cases}$$

q=2
a1 =1

Example 1.

Binary Trees

Each nodes has at most 2 children.



Q1 Suppose a binary tree has height h.
What is the maximum number of nodes in the tree?

$2^0 + 2^1 + 2^2 + 2^3 \cdots 2^h = 2^{h+1} - 1$

Q2 Suppose we know a binary tree has n nodes.
What is the minimum height of tree?

max height: n-1

min height: If the height of the tree is h

$n \leq 2^{h+1} - 1$

$n+1 \leq 2^{h+1}$

$\log_2(n+1) \leq \log_2(2^{h+1})$

$\log_2(n+1) \leq h+1$

$\log_2(n+1) - 1 \leq h$

Example 2:

Analyzing the # of iterations.

let $b > 1$ be a postive integer.

```
Loop(n):
while n > 1
|   n ← n/b
endwhile
```

$n \to \frac{n}{b} \to \frac{n}{b^2} \to \frac{n}{b^3} \to \cdots \to \frac{n}{b^k}$

Find the smallest integer k s.t. $\frac{n}{b^k} \leq 1$

$\frac{n}{b^k} \leq 1$

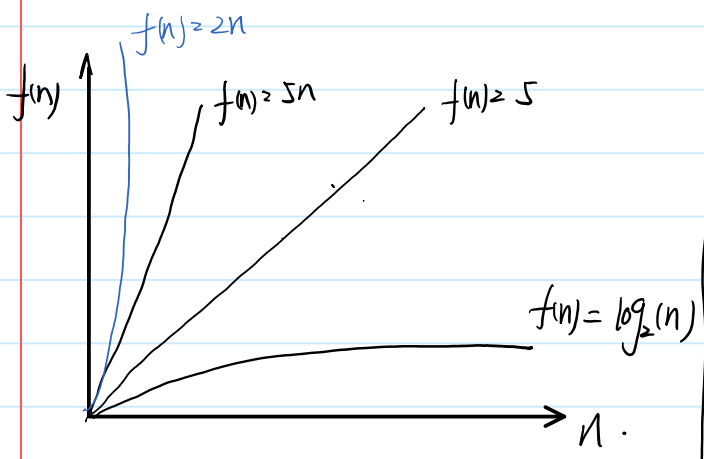
$n \leq b^k$

$\log_b n \leq k$ $\quad k = \lceil \log_b n \rceil$

3

2017年1月23日 15:57

The functions $\log_2 n$, n, $5n$, 2^n



$$(\log_a x)' = \frac{1}{x \ln a}, (\ln x)' = \frac{1}{x}$$

$$(a^x)' = a^x \ln a, (e^x)' = e^x$$

Growth rates.

Compare growth rate of $f(n)$ & $g(n)$

Consider the ratio $\lim_{n\to\infty} \frac{f(n)}{g(n)}$

- n vs. $5n$ $\lim_{n\to\infty} \frac{n}{5n} = \frac{1}{5}$
- $\log_2 n$ vs. n $\lim_{n\to\infty} \frac{\log_2 n}{n} = \lim_{n\to\infty} \frac{\frac{1}{n \log_2 e}}{1} = \lim_{n\to\infty} \frac{1}{n \cdot \log_2 e}$

  Lorbita rule, derivate $= 0$

  L'Hospital's rule
- n vs. 2^n $\lim_{n\to\infty} \frac{n}{2^n} = \lim_{n\to\infty} \frac{1}{2^n \log_e 2} = 0$

$\log_2 n << n, 5n << 2^n$.

2017年1月23日 15:57

Asymptotics notation

$\log n$, 5, $5n$, 2^n

U.B.

DEF: Let $f(n)$ and $g(n)$ be functions of n. We say that $f(n)$ is $O(g(n))$ if there exists constant $c>0$ and $n_0 \geq 1$ s.t. $f(n) \leq c \cdot g(n)$ where $n \geq 0$.

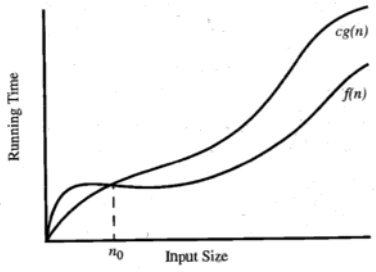


Figure 1.5: Illustrating the "big-Oh" notation. The function $f(n)$ is $O(g(n))$, for $f(n) \leq c \cdot g(n)$ when $n \geq n_0$.

("the growth rate of $f(n)$ is bounded above by the growth rate of $g(n)$")

Ex1:

$8n^5 + 32n^4 + 1000$ is $O(n^5)$

$8n^5 + 32n^4 + 1000 \leq c \cdot n^5$ when $n \geq n_0$

$8n^5 + 32n^4 + 1000 \leq cn^5 + 32n^5 + 1000n^5$

$n \geq 1$

Let $c = 1040$ and $n_0 = 1$.

Ex2: $100n$ is $O(2^n)$.

$100n \leq c \cdot 2^n$ when $n \geq n_0$

$c = 1$ $n_0 = 10$

5

2017年1月23日 15:57

$f(n)$ is $O(g(n))$ U.B.

$f(n)$ is $\Omega(g(n))$ L.B.

$f(n)$ is $\theta(g(n))$ "same" growth rate.

Relatives of Big O notation

(1) $f(n)$ is $\Omega(g(n))$ is $O(f(n))$ L.B.

the growth rate of $f(n)$ is bounded below by the growth rate of $g(n)$

(2) $f(n)$ is $\theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$ same growth rate

EX 3 $8n^5 + 32n^4 + 1000$ is $\Omega(n^5)$

show n^5 is $O(8n^5 + 32n^4 + 1000)$

find $c=0$, $n_0 \geq 1$,

so that $n^5 \leq c \cdot (8n^5 + 32n^4 + 1000)$.

$c=1$, $n_0=1$.

$\therefore 8n^5 + 32n^4 + 1000$ is $\theta(n^5)$

Tricks:

When comparing growth rate of function $f(n)$ & $g(n)$

consider $\lim_{n\to\infty} \frac{f(n)}{g(n)}$

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \text{then } f(n) \text{ is } O(g(n)) \\ k, \text{ a constant} & f(n) \text{ is } \theta(g(n)) \\ +\infty & f(n) \text{ is } \Omega(g(n)) \end{cases}$

THM 1.7

(1) $\log^d n$ is $O(n^k)$ for any constants $d>0$ and $k>0$

2017年1月23日 15:57

Ex $\log^{60} n$ is $O(n^{0.00001})$

(2) n^k is $O(a^n)$ for any constants $k>0$, $a>1$.

Ex n^{500} is $O((1.01)^n)$

Some applications.

1. A binary tree with height h has at most 2^h-1 nodes

has $O(2^h)$ nodes

If a binary tree has n nodes, it is height is at least $\log_2(n+1)-1$

$\Omega(\log n)$

2. for a positive integer n, let $f(n) = 1+2+3-\cdots+n$.

what is $f(n) = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$ $\therefore$ $f(n)$ is $O(n^2)$

```
S ← 0
for i=1 to n
|   for j=1 to i
|   |   S ← S+j
|   endfor
endfor
```

# of addition steps $= 1+2+3-\cdots n = \frac{n(n+1)}{2}$ is $O(n^2)$

$1^2+2^2+\cdots n^2 = \frac{n(n+1)(2n+1)}{6}$ is $O(n^3)$

For constant $k>0$

$1^k+2^k+\cdots n^k$ is $O(n^{k+1})$

2017年1月23日 15:57

3. Geometric series. (recursion)

For positive integer n. Let $g(n) = 1 + a + a^2 + \cdots a^n$ for $a > 0$

Growth rate.

FACT, when $a = 1$ $g(n) = n+1$ $g(n)$ is $O(n)$

$a \neq 1$ $g(n) = \frac{a^{n+1}-1}{a-1}$

$a > 1$ $\frac{a^{n+1}-1}{a-1} \approx \frac{a^{n+1}}{a} = a^n$ $g(n)$ is $O(a^n)$

$a < 1$ $\frac{a^{n+1}-1}{a-1} = \frac{1-a^{n+1}}{1-a} = \frac{1}{1-a}$. $g(n)$ is $O(1)$

Final notes

1. In CS, the following terms are using informally.
We say that function $f(n)$ is

- logrithmic if $f(n)$ is $O(\log^k n)$ for some constant $k \geq 1$
- Polynomial if $f(n)$ is $O(n^k)$
- exponential if $f(n)$ is $\Omega(a^n)$ for $a > 1$.

EX. $\frac{n}{\log n}$ is $O(n)$

$n!$ is $\Omega(a^n)$ ↓ for some $a > 1$

$n^{2.5}$ is $O(n^3)$

$(\log\log n)$ is $O(\log n)$

8

2017年1月23日 15:57

2. When comparing growth rate of function, ignore coefficient and constant number of lower odered terms.

Ex. ~~1000~~ 2^n Vs. $3^n + 500n^2 + 8$

$\theta(2^n)$ $\theta(3^n)$

$$\lim_{n\to\infty} \frac{2^n}{3^n} = \lim_{n\to\infty} \left(\frac{2}{3}\right)^n = 0$$

3^n is much faster.

Week2

2017年1月23日 15:57

$O(1)$ — growing at most as fast as a constant

$f(n) = 5$; $f(n) = 1/n$

☆ $O(n)$ — linear in n

$f(n) = 5n + 6\sqrt{n} + \log 1$ is $O(n)$ $\lim_{n\to\infty} \frac{5n+6\sqrt{n}+\log 1}{n} = 5$

$O(n \log n)$

$O(n^2)$ — quadratic in n.

$O(2^n)$

Big O Notation can hide big constants & significant lower order terms.

Algorithm Design & Analysis

Two aspects that's important to us.

(1) correctness of algorithm. } Proofs
↳ all inputs

(2). Running time of algorithm. }
↳ efficient/scalable. }

Framework.

(1) algorithm will be describle using pseudo-code
- a mix of English & actual code.
- readable

(2) The algorithm's running time will be based on the number of primitive operations/steps it performs. in the worst case.

primitive operations, take $O(1)$ time.

Arithmetic operations, $+, -, \times, \div$

Assigning a value to a variable, $x \leftarrow 0$

2017年1月23日 15:57

comparing two number $x>y$; $x=y$

indexing an array, $A[i]$

$\Rightarrow$ need to count the # of primitive operations.

$\Rightarrow$ a function of the input size (n)

$\Rightarrow$ assume worst case behaviour. ☆

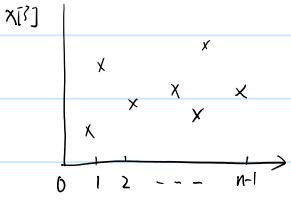
* average case

~~best case~~

(3) The algorithms running time will be expressed using the Big O notation.

EX. Prefix Averages

Let $x[i]$, $i=0$ to $n-1$ denote the price of a stock at day i



$$A[i] = \frac{x[0]+x[1]+\cdots x[i]}{i+1}, \quad \text{for } i=0 \text{ to } n-1$$

$$A[i+1] = \frac{x[0]+x[1]-\cdots x[i]+x[i+1]}{i+2}$$

GOAL: Given $x[0,\cdots n-1]$, compute $A[0,\cdots n-1]$

Prefix averages $(X[0\cdots n-1])$

Create an array $A[0\cdots n-1]$, — $c_1 \cdot n$ $\theta(n)$

for $i=0$ to $n-1$

| $A[i] \leftarrow \frac{X[0]+X[1]+\cdots+X[i]}{i+1}$ — $\sim c_2 \cdot i$ $\theta(i)$ $\Rightarrow \sum_{i=0}^{n-1} c_2 \cdot i$

EndFor

Return(A) — $\theta(1)$

$= c_2(0+1+2\cdots n-1)$

$= c_2 \frac{(n-1)n}{2}$

is $O(n^2)$ / $\theta(n^2)$

The running time is $\theta(n^2)$ or $O(n^2)$

2017年1月23日 15:57

Prefix Average 2 (X[0..n])

Create an array A[0..n-1] —— $\theta(n)$

Sum ← 0 —— $O(1)$

for i=0 to n-1 } $O(1)$ per iteration, $O(n)$ for all iteration.

  Sum ← Sum + X[i]

  A[i] ← Sum / (i+1)

Endfor

Return (A) —— $O(1)$

Running time : $O(n)$

The Stable marriage problem.

Each has a preference list

n men, n women → n pairs

GOAL: pair up the men & women so that they will not tempted to leave their partners.

1960's Gale & Shapley

1980's Alvin Roth.

DEF: A stable matching M consist of n pairs of men & women.

st.

(1) each person is part of exactly one pair.

(2) there is no pair (m,w) st. they prefer each other than their partners in M. → blocking matching

EX:

m_1: w_2 w_4 w_1 w_3	w_1 m_2 m_1 m_4 m_3
m_2 w_3 w_1 w_4 w_2	w_2 m_4 m_3 m_1 m_2
m_3 w_2 w_3 w_1 w_4	w_3 m_1 m_4 m_3 m_2
m_4 w_4 w_1 w_3 w_2	w_4 m_2 m_1 m_4 m_3

Q: Is (m_1,w_2) (m_2,w_3), (m_3,w_2) (m_4,w_4) a stable matching?

Not stable!

2017年1月23日 15:57

CLAM #1. The algorithm terminates
CLAM #2 The algorithm output a stable matching

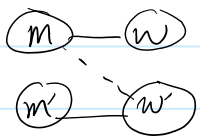
observation — (i) The women m proposes to gets worse & worse according to His preference List.
In particular, he never propose a women twice

(ii) Once a woman is engaged., she remains engaged.
The man w engaged to gets better & better according to her PL.

(iii) A person (through out the algorithm) is free or matched. to one person only.

Proof of the claim 1.
The algorithm will terminate when all men are matched.
Consider an arbitary man m.
— at some point, m is permanently matched to some woman.
✗ — he is rejected by all the women & so in never matched.
m, w_1, w_2, ... w_n. n women ← n-1 women.
⇒ ALL women had partners
∴ ⇒⇐

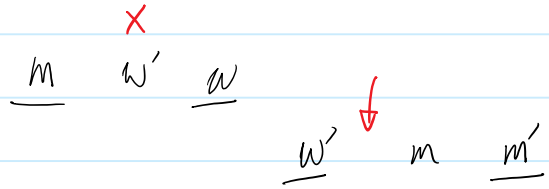
Proof of the claim 2.
Suppose M = {—, --, ···→} is output.
M is matching
Assume M as a blocking pair. (m, w')
In M, (m, w) is a pair (m', w') is a pair



4

2017年1月23日 15:57

Since (m, w') is a blocking pair.



— m propose to w', but was eventually rejected for a better partner.

—— w's partners gets better & better.

∴ w' can't be matched to m' ⇒⇐

∴ M is a stable matching.

5

2017年1月23日 15:57

1. a. Loop 3, the running time is $O(n^2)$

Loop 4, $\sum_{i=1}^{2n} O(1) = O(\sum_{i=1}^{2n} 1) = O(\frac{(2n+1)\times 2n}{2}) = O(n^2)$

the running time is $O(n^2)$.

b. Create matrix A,

```
for k ← 1 to n-1 do
    for i ← 1 to n-k do
        j = i + k
        A[i][j] ← 0
        for x ← i to j-1 do
            A[i][j] ← A[i][j] + A[i][x] × A[i+j-x, j]
        end for
    end for
end for
```

$(n-1)+(n-2)\cdots+1$

$O(\frac{n\times(n-1)}{2}) = O(n^2)$

$O(j-i) = O(n)$

$O(n^2) \times O(n) = O(n^3)$

∴ the running time is $O(n^3)$.

2. Create matrix A[i][j]

```
Total ← 0
i ← n-1 , j ← 0
while ( i > 0 and j < n) do
    if A[i][j] = 1 then
        j ← j+1
    else
        Total ← Total + j
        i ← i-1
    end if
end while
```

$O(n)$

$O(n)$

$O(n) + O(n) = O(2n) = O(n)$

∴ the running time is $O(n)$.

3.

Set. m_i as men m

m_s as men m'

w_r as women w

Data structures:

stack S —— store free man

array R —— store index of first women has not reject him for each man

array A —— store matched man for each women.

matrix Q' —— show ranked preference for each women.

```
Create Q'
for i ← 1 to n do
    for j ← 1 to n do
        k ← Q[i][j]
        Q'[i][r] ← j
    end for
end for
```

```
create S, R, A
for i ← 1 to n do
    S[i] ← m_i
    R[i] ← 1
    A[i] ← 0
end for
```

(ii) running time of setting Q' is $O(n^2)$.

Other settings takes $O(n)$,

there are no more than n^2 proposals during a run of algorithm, and each proposal can be performed in $O(1)$ time.

∴ the running time of Gale-Shapley algorithm is $O(n^2)$

It's best possible implemention, because I use a matrix to store links of women, men, and her preference, which is a strategy of trading space for time. So we don't need to traverse the preference list.

Algorithm:

```
While S[1] != null do
    m_i ← pop S
    w_r ← R[m_i]
    if A[w_r] = 0 then
        A[w_r] ← m_i
    else
        m_s ← A[w_r]        // m' matched to w
        if Q'[w_r][m_i] < Q'[w_r][m_s] then
            A[w_r] ← m_i
            Push m_s to S
        else
            R[w_r] ← w_r + 1
            Push m_i to S
        end if
    end if
end while
```

4.

Create matrix P' for ranked preference of man

Because $M[i]=r$, $M[i]$ means current women of a given man,

So, create array M' to store current man of a given women.

```
for i ← 1 to n do
    for j = 1 to n do
        r ← P[i][j]
        P'[i][r] ← j
    end for
end for
```

$O(n^2)$

```
for i ← 1 to n do.
    j ← M[i]
    M'[j] ← i
end for
```

$O(n)$

Algorithm:

```
stable match = true
for i ← 1 to n do
    for r ← 1 to n do
        if P[i][r] < P'[i][M[i]] and Q'[r][i] < Q[r][M[r]] then
            stable match = false
            store blocking pair (m_i, w_r)
        end if
    end for
end for
    if stable match = true then
        print "yes"
    else
        print "no"
        print blocking pair (m_i, w_r)
end if.
```

$O(n^2)$

The algorithm try to traverse all the pairs to compare with current match, and find blocking pair, which is correct.

Overall, the running time is $O(n^2)$

Week3

2017年1月23日 15:57

```
Loop3 (n)
    s ← 0
    for i = 1 to n^2
        for j = 1 to i
            s ← s + i
```

inner loop: $O(n^2)$ worst case; $O(i)$

outer loop: $O(n^4)$; $i = 1$ to n^2

Running time $\sum_{i=1}^{n^2} O(1) = O\left(\sum_{i=1}^{n^2} i\right)$

$= \frac{n^2(n^2+1)}{2} = \frac{n^4+n^2}{2}$

Chapter 2. Basic Data Structures

ADT (Abstract Data Types)

(i) an interface
- describe the methods/operations that can be used to access the data.

(ii) the implemetation
- includes the internal representation of the data (together with the algorithms used to implement the methods of ADT)

1. Stacks

- is a container of objects that are added & removed using the LIFO Principle.

The method,

push(o) = insert "o" at the top of the stack

pop = remove object at the top of the stack

size(), isEmpty(), top().

2

2017年2月6日 15:55

Implementation.

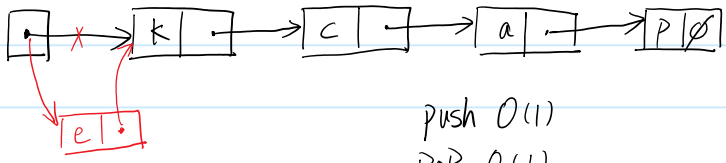
1. Array - based.
   - an array S of size N is created
   - an index "t" is used to keep track of the top of the stack; S[t]

   Push } O(1)
   Pop }

2. Linked - list based
   - A Linked list L is created.

the — header of L points to the top of the stack

EX.



push O(1)
pop O(1)

pros/cons

1. array size is fixed
   - may need to resize
2. For Link-list pointers occupied half of the space.

---

2. QUEUES
   - like stacks except the object are added & removed using FIFO principle.

TWO- methods. (2 access point)

enQueue(o) — insert "o" at the rear (back) of the Queue.
deQueue(o) — removes the object from the "front" of the Queue.

size()
isEmpty()
front()

3

2017年2月6日 15:58

Implemetation..

1. array-based.

- an array Q of size N is created.

~~in~~ dot f keeps track of the Queue f.

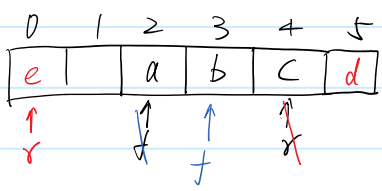
index r ... rear of the Queue.

Q(F)

Q(R)

"circular array"

Ex: Q:



enQueue (d) } O(1)

enQueue (e) }

deQueue —— O(1)

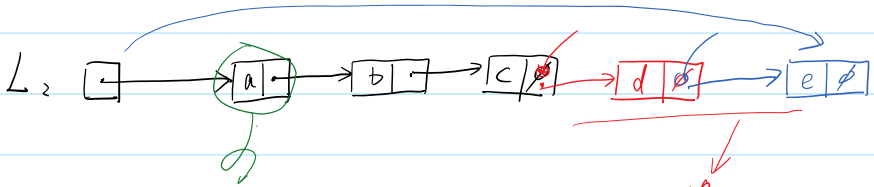
2. Linked-list

— A Linked list L is created.

— the header points to the "front" of the Queue.

So the tail corresponds to the "rear" of the Queue.

L:



dequeue.

O(1)

enQueue.

O(n)

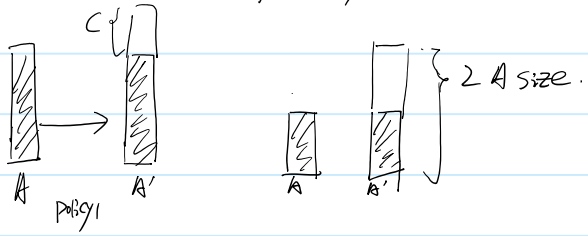
O(1)

Resizing Arrays: (Amortized Analysis)

Policy 1: Increase the array A by a constant C

Policy 2: Double the size of array A.

Resize(A):



4

2017年2月6日 15:58

Policy 1.

Create A' : $O(A.size + c)$ → $O(A.size)$

copying : $O(A.size)$

Policy 2.

Create A' : $O(A.size + A.size)$ → $O(A.size)$

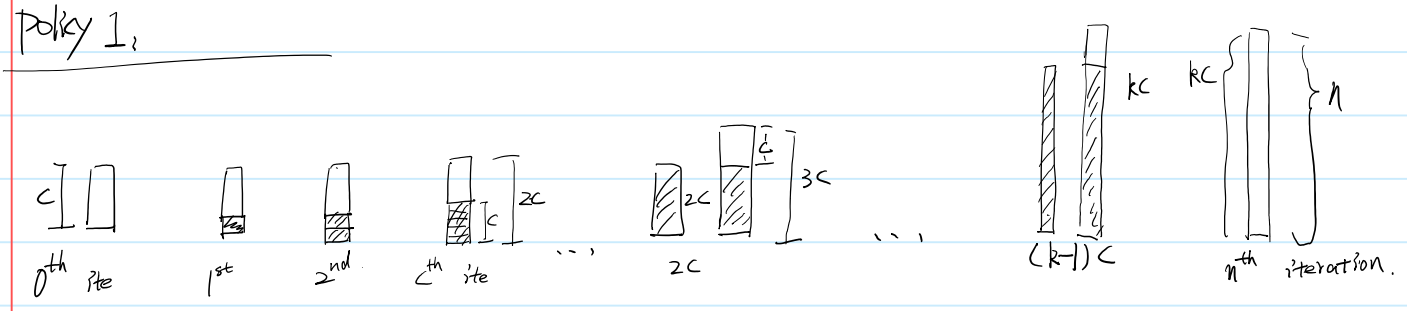
copying : $O(A.size)$

Start with a array of size 0 or 1

GOAL : insert n objects.

Assumption. Resize immediately when an array becomes full.

Policy 1:



$kc \geq n$

$k \geq \frac{n}{c}$

$\therefore k = \lceil \frac{n}{c} \rceil$

How much work?? (running time).

$\theta(n) + O(c+2c+\cdots kc)$

↑ adding

$= \theta(n) + O(c(1+2+\cdots k))$

$= \theta(n) + O(c \cdot k^2)$

$= \theta(n) + O(c \cdot \frac{n^2}{c^2}) = \theta(n) + O(\frac{n^2}{c}) = O(n^2)$

dominant term

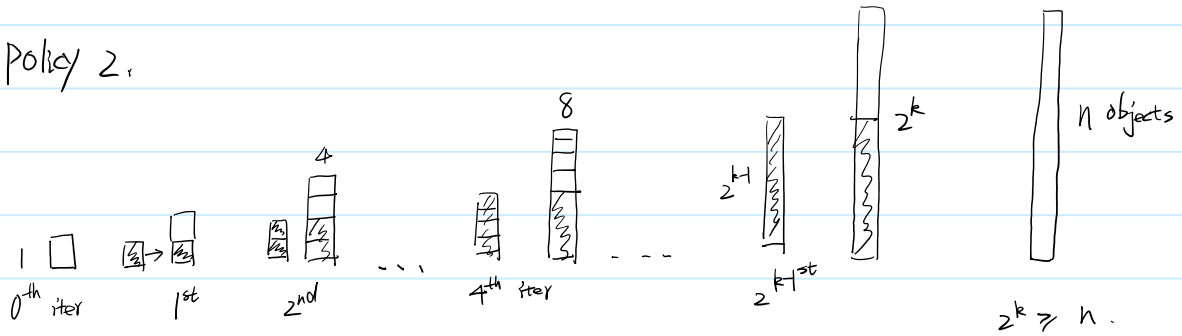
$O(n^2)$

→ $O(n)$ per object (n objects)

5

2017年2月6日 15:56

Policy 2.



$2^k \geq n$.

$k \geq \log_2 n$

$k \geq \lceil \log_2 n \rceil$

$\Theta(n) + O(1 + 2^1 + 2^2 \cdots + 2^{k-1})$ ($2^k - 1$ terms)

$\Theta(n)$: adding

$= \Theta(n) + O(2^k) = \Theta(n) + O(c 2^{\log_2 n}) = \Theta(n) + O(n) = O(n)$

$O(1)$ per object

6

2017年2月8日 15:22

3. VECTORS

— an ADT that behaves like arrays

DEF: Given an ordered sequence of n objects,
the rank of an object O is the number of objects ahead of O.

EX: abcde rank(a) = 0 rank(e) = 4.

A vector is an ADT where the objects are linearly ordered & access by rank.

	Array	Linke list
elemAtRank(r)	O(1)	O(r)
replaceAtRank(r,e)	O(1)	O(r)
insertAtRank(r,e)	O(n-r+1)	O(r)
removeAtRank(r)	O(n-r+1)	O(r)
Size( )		
isEmpty( )		

Implementations,

(i) array

— use an array A to store the elements where A[i] contains elements at Rank i

(ii) linked-list

— use an Link-list L to store the elements where i^{th} node contains elements at Rank i-1

7

2017年2月8日 15:22

4.

LISTS

— an ADT that behaves like doubly-linked lists.

— each object is stored in a "position"
& the positions are ordered linearly.

— objects are accessed via their positions

before (p)
after (p)
replaceElement (p,e)
swapElement (p,q)
insertBefore (p,e)
insertAfter (p,e)
remove (p)

} O(1)

Implementations.

(i) Doubly linked list.

---

5. TREES

— the internal structure is hierachical.

because (i) the data is naturally hierachical

(ii) it is best accessed if given a hierachical structure.

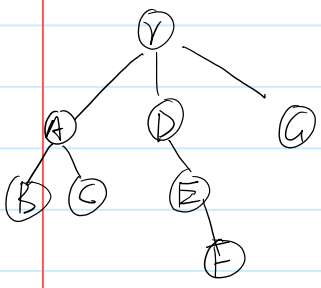
A TREE ADT T stores elements in a parent-child relationship.

- T has a special node r, the root of T
- For each node $v \neq r$ v has a parent node, parent (v), v.parent.

8

2017年2月8日 15:22

Teminology.

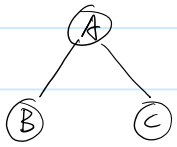


— A is the parent of B & C.
A has two children B & C
B & C are siblings.

— Node with no children are external nodes.
otherwise are internal nodes.

— E has ancestor E D, R
has decendants E & F

Subtree at A



∴ ordered tree
— children of nodes have an ordering.

TREE ADT Methods.

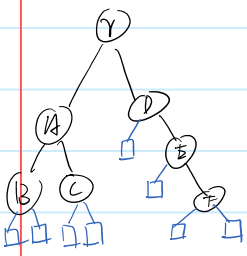
root ( )
parent (v)
children(v)
leftchild (v)
right child (w)
sibling (v)

isInternal (v)
isExternal (v)
isRoot (v)
SwapElements (v, w)
replaceElements (v, e)

size ( )
element ( )
positions ( )

Binary tree.

— is an ordered tree where each node has at most two children: left & right child.



Proper Binary Tree
— Every internal node have exactly two children.

2017年2月8日 15:22

Tree Traversals

— algorithms that visit every node of the tree to do some processing.

1. Pre-order Traversal

— process a node before any of its descendants.

```
PreOrder (T, v)
Process v
for each child w of v.
    | preOrder (T, w)
end for
```

2. Post-Order Traversal

— process all the descendants of a node before the node.

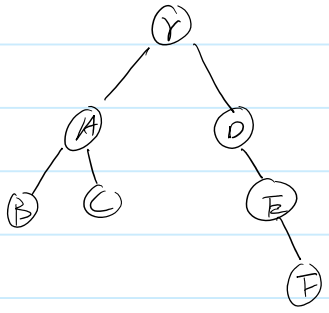
```
Post Order (T, v).
for each child w of v
    | PostOrder (T, w)
end for
Process v.
```

beginning T root.

3. In-Order Traversal (Binary Tree).

```
In-Order (T, v)
if leftchild exists.
    InOrder (T, v, leftchild)
Process V.
if V Rightchild exists.
    InOrder (T, v, Rightchild)
```

Ex:



PreOrder (T, Y): Y A B C D E F

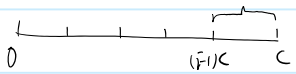
PostOrder (T, Y): B C A F E D Y

InOrder (T, Y): B A C Y D E F

Week4

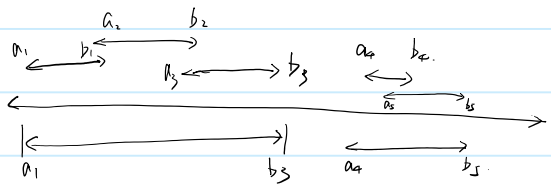
2017年2月13日 15:26

1. Binary Search (low, high, x)



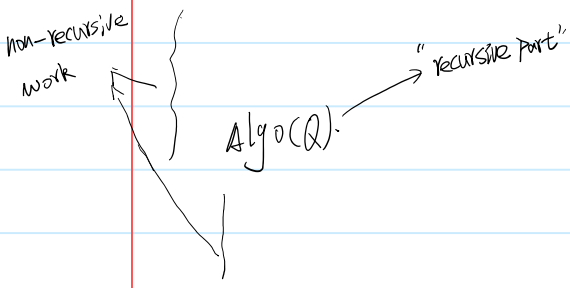
O log (high − low)

4. (2-23) Input: n intervals (a_i, b_i) for $i = 1$ to n



Polynomimal time.
$O(n \cdot \log n)$

Algo(P)



PreOrder (T, v)

$O(1)$ → Process v

$O(C_v)$ → for each child w of v.

| PrOrder (T, w)

end for

$O(C_v)$

↳ non-recursive work at preorder (T, v)

running time

$$\sum_{v: v \text{ is a node in } T} O(1 + C_v)$$

$$= O\left(\sum_{v} (1 + C_v)\right)$$

$$= O\left(\sum_{v} 1 + \sum_{v} C_v\right)$$

n: # of nodes in T

$$= O(n + (n-1))$$

$$= O(n)$$

* Assume processing v takes $O(1)$ time.

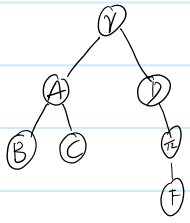
PreOrder, PostOrder, InOrder takes $O(n)$ time.

2017年2月13日 15:26

Parameter associated with a tree.

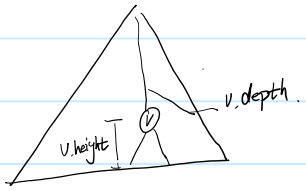
DEF. The depth of node v in T is equal to the # of ancestor of V, excluding itself

The height of T is the maximum depth of a node in T.



depth(B) = 2

height(T) = 3.



depth(T,V)

```
if V = T.root
|  Return (0)
else
   Return (1+ depth(T, V.present))
```

O(depth(v))

PreOrder

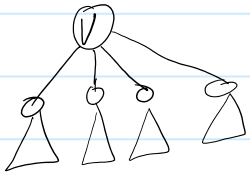
AllDepth(T,v)

```
if V = T.root
|  V.depth ← 0
else
   V.depth ← 1+ V.parent.depth
   for each child w of v.
   |  AllDepth(T,w)
   End for
```

Run AllDepth(T, T.root)

Running Time O(n)

AllHeight(T,V) → T.root



V.height = 1+ max{w.height}

```
if isExternal(T,V)
|  V.height ← 0
else
   h ← 0
   for each child w of v.
   |  AllHeight(T,w)
   |     ← max(h, w.height)
   endfor
   V.height ← 1+h.
```

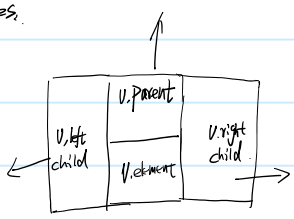
O(n)

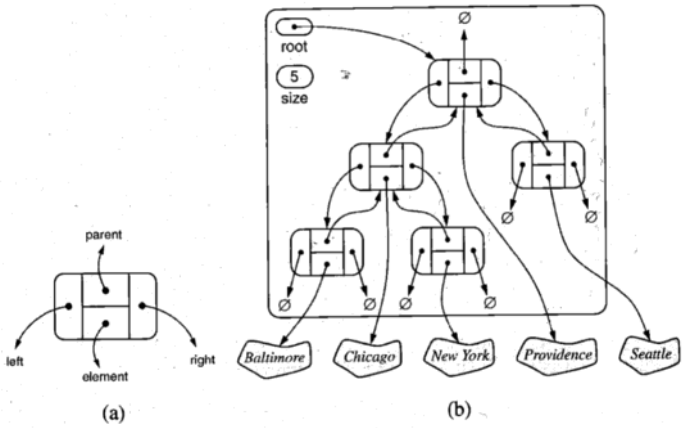
3

2017年2月13日 15:27

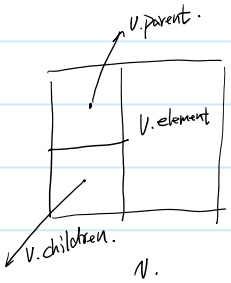
Implementations of trees:

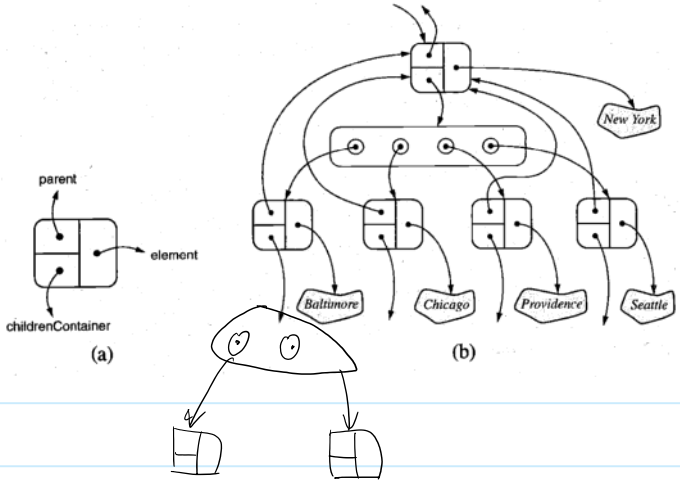
Binary trees:





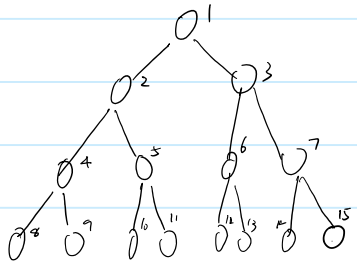
General Trees:





Binary Tree:

— an array-based implementation



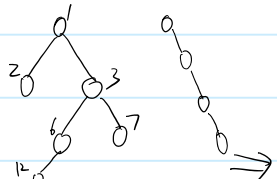
For each node v.

let $l(v)$ denotes its "number"

If $u = v.leftchild$, $l(u) = 2 \times l(v)$

$= v.rightchild$. $l(u) = 2 \times l(v) + 1$

$w = v.parent$ $l(w) = \lfloor \frac{l(v)}{2} \rfloor$



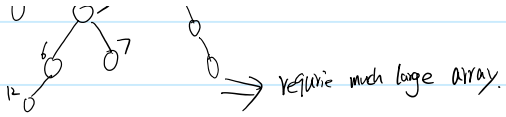
⇒ require much large array.

Suppose T has n nodes. Create an array $S[1 \ldots n]$ where node v is associated with $S[l(v)]$

- root [T] — $S[1]$
- isInternal(v) $S[2 \cdot l(v)], S[2 \cdot l(v)+1]$
- isExternal(v)
- isParent(v) $S[\lfloor \frac{l(v)}{2} \rfloor]$

leftchild(v) $S[2 \cdot l(v)]$

rightchild(v) $S[2 l(v)+1]$


0
6
7
12
0
0
⇒ require much longe array.

4

2017年2月15日 15:27

PRIORITY QUEUES

· Our data consist of (key, elements) pairs.

— the keys are linearly ordered. [removeMax( )]

· We want to support two methods. insertItem(k,e), removeMin( ).

→ Min Priority Queue

→ Max Priority Queue

Application:

(i) processing jobs

keys — Importance of the job

— Due date

— length of a job

Policy: process a job based on earliest due day / important etc.

(ii) Sorting

Suppose L is a list of n numbers that needs to be sorted.

Ex: 35, 2, 100, 20

(35,35) (2,2) (100,100) (20,20)

— Insert n numbers into a min-PQ

— Do n removeMins.

Implementation:

(i) use an array or doubly linked list.

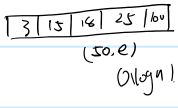
insertItem —— O(1)

removeMin —— O(n)

(ii) Use a sorted array or sorted doubly-linked list.

insertItem —— O(n)

removeMin —— O(n) O(1)



(50,e)

O(log n)

5

2017年2月15日 15:27

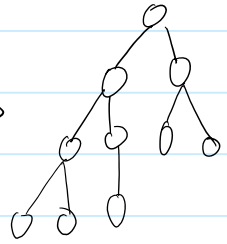
3. Heaps

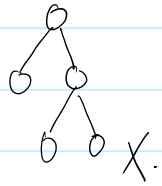
[Trees with extra rules]

A Heap is a complete binary tree. Crooks are attached. (top → bottom / left to right)

full binary tree.

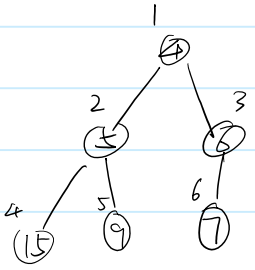
the items satisfied the heap order property:





for each node. v≠root,

key(v) ≥ key(v.parent)



key(Root) is always a min key.

1	2	3	4	5	6
4	5	6	15	9	7

Some advantages of Heaps:

(i) Finding min key is easy → Root !!

(ii) Easy to implement

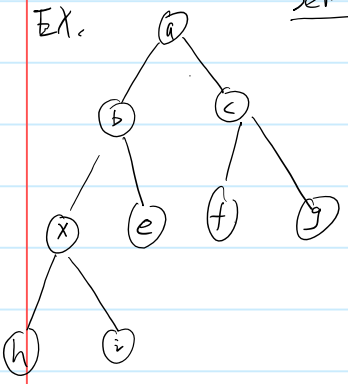
— use array representation of trees.

(iii) if the heap has n items, the height of the tree is O(log n)

$\lceil \log_2(n+1) \rceil$

"Semi-Sorted"

Ex.



Which keys are ≥ x ? a, b.

≤ x ? h, i

MGHH6

2017年2月15日 15:27

Convention: Let last be a reference to be last node in T.

Methods:

1. insertItem (k, e)

— create a new note at T[last+1] place item (k, e)

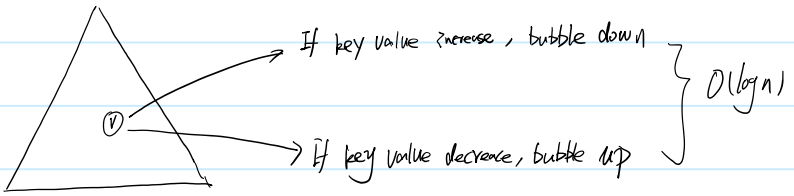
"up" — "heap Bubble" until heap order property is restored. $O(\log n)$

2. removeMin

— return the root's item. & Replace it with the item in T[last], last ← last − 1

— Down heap bubble until the heap-order property is restored. $O(\log n)$

3. replaceKey (v, k)



4. removeItem (v) — replace with item with T[last]

and bubble up/down $O(\log n)$.

Heap Sort · (Sort n numbers)

— Insert all n numbers into a heap. $O(n \log n)$

— perform n removeMins. $O(n \log n)$

$O(n \log n)$

"Inplace"

Inplace HeapSort ·

Assume that the numbers to be sorted are already in array.

EX:

30	88	21	5

STEP1: From left to right,

Insert each number in a PQ.

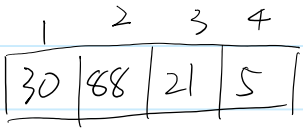
STEP2. From left to right ·

delete the smallest element. Swapping it with the last element.

Result: Numbers are sorted from largest to smallest.

7

2017年2月15日 15:27



8

2017年2月15日 15:47

9

2017年2月15日 15:47

Week5

2017年2月20日 14:55

Priority Queues

Data: (key, element) pairs.

Insert Item.

RemoveMin (RemoveMax)

Heap

- complete Binary tree

no holes in middle.

height O(logn)

stores in an array

- key(v) ≥ key(v.parent), ∀ v ≠ Root

Root is always smallest key.

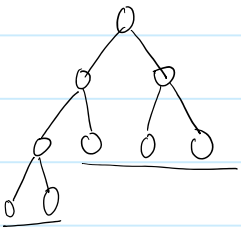
{ Insert Item. / RemoveMin (RemoveMax) } O(logn) Bubble up / Bubble down.

Bottom-up heap construction

n items already stored in an array (i.e. in memory)

⇒ Do n insertItems: O(nlogn)

O(n) time



# of leafs in complete binary tree is ≥ internal nodes.

O(n/2 logn)

Bottom-up construction

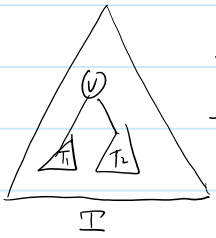
- post order traversal

① First, stores the items in I (1...n)

② Run BuildHeap in a post-order traversal manner.

2

2017年2月20日 14:55



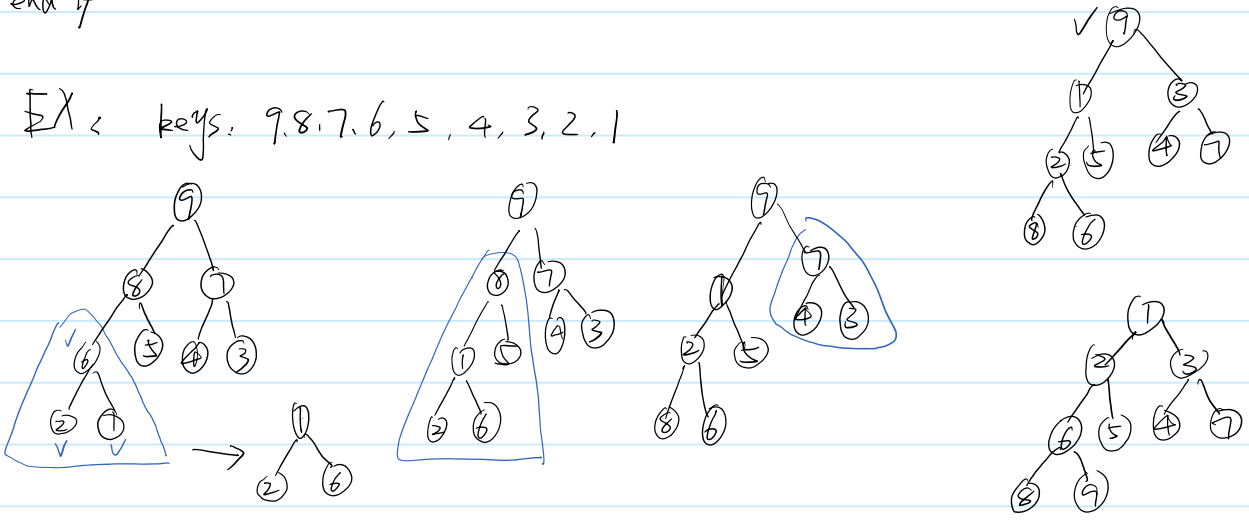
T

- Make T_1 into a heap
- Make T_2 into a heap
- Merge them by making v the parent
- If v.T > v.leftchild.key or v.rightchild.key
  bubble down !!

BuildHeap (T, v)

if v is an external node
  Return
else
  BuildHeap (T, v.leftchild)
  if v.rightchild exists.
    BuildHeap (T, v.rightchild)
  endif
  if v.key > v.leftchild.key or v.key > v.rightchild.key
    ⇒ DownHeapBubble (T, v).
  end if
end if

EX: keys: 9, 8, 7, 6, 5, 4, 3, 2, 1



CLAIM: BuildHeap (T, Root) turns T into a heap in O(n) time.

(a lot of work of very little nodes) → DownHeapBubble.

2017年2月20日 14:55

Chapter 3.

ORDERED DICTIONARIES.

Data: (key, element) pairs.

keys have a linear ordering

Methods: findElement(k) closestKeyBefore(k)

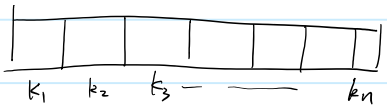
insertItem(k,e) closestElementBefore(k)

removeItem(k) closestKeyAfter(k)

closestElementAfter(k)

Implementations:

1. Sorted array.



findElement(k) —— Binary Search $O(\log n)$

insertItem — $O(n)$

removeItem — $O(n)$

closestKeyBefore } $O(\log n)$

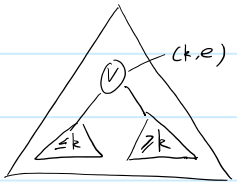
closestKeyAfter }

---

2. Binary Search Tree. —— Binary tree with extra requirements.

Binary Tree —— structure.

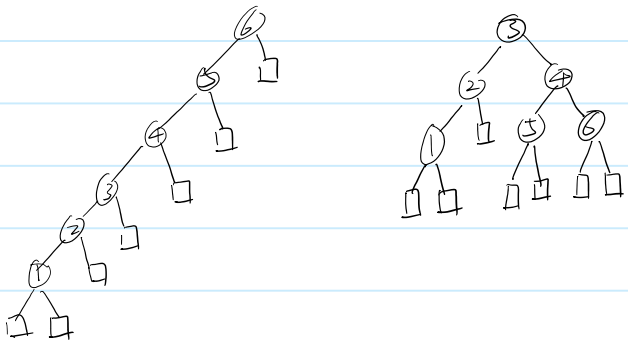
(where each internal node stores an item (k,e)



4

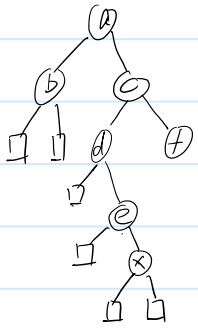
2017年2月20日 14:55

Ex. keys: 1, 2, 3, 4, 5, 6.



Notes:

1. consider a BST



Assume — letters are keys.

— keys are distinct.

< x: b, a, d, e

> x: c, f

x in the right subtree of b

project down element

b a d e c f

2. In Order Traversal.

will output keys in sorted order.

Methods:

1. Find element (k)

Tree Search (k, v)

If v is an External node.

Return v

if v.key = k.

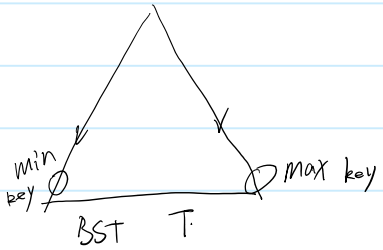
Return v.

if v.key < k

| Return TreeSearch (k, v.right child)

else

Return Tree Search (k, v.left child)



O(T.height) → O(n)

5

2017年2月20日 14:55

2. InsertItem (k, e)

insertItem (k, e)

w ← TreeSearch (k, T.root)

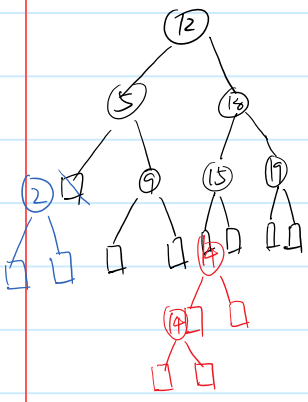
while w is not an external node.

| w ← TreeSearch (k, w.leftchild or rightchild)

end while

Store (k, e) at w & create two external nodes.

EX.



Insert 2. 14, 14

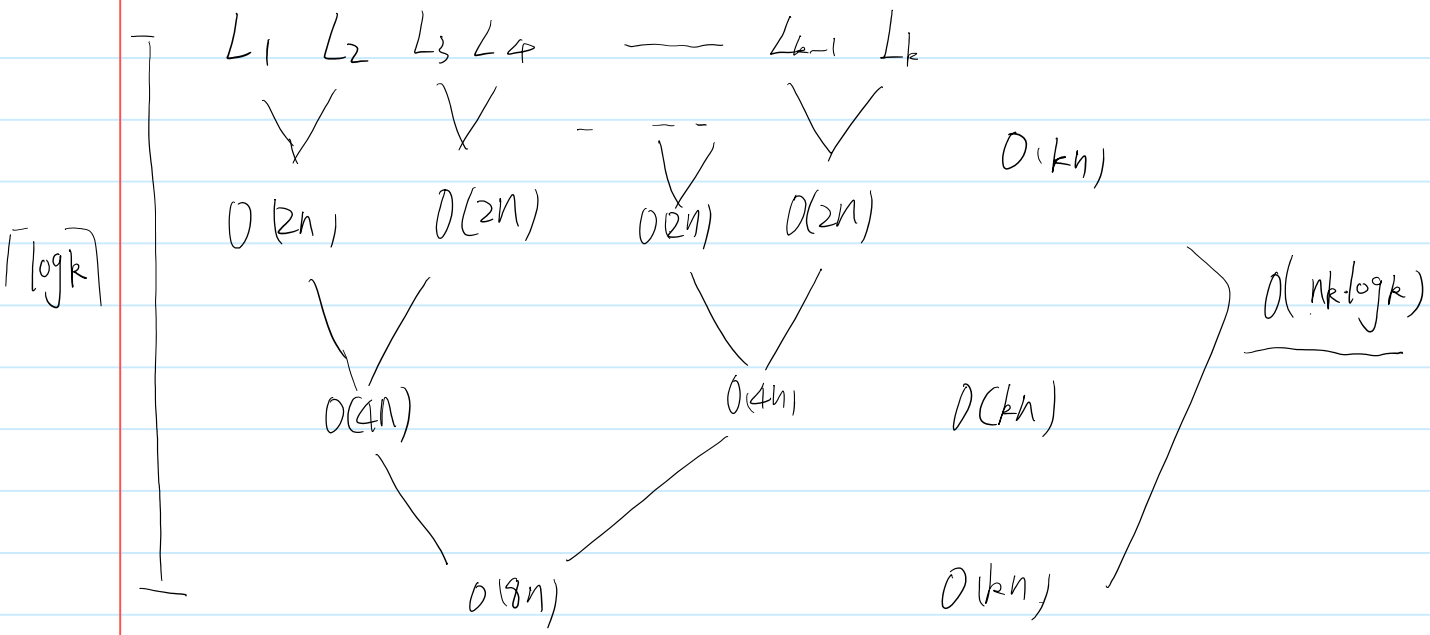
O(T.height)

6

2017年2月22日 14:55

Mergesort combine 2 sorted list

$O(L_1.size + L_2.size)$

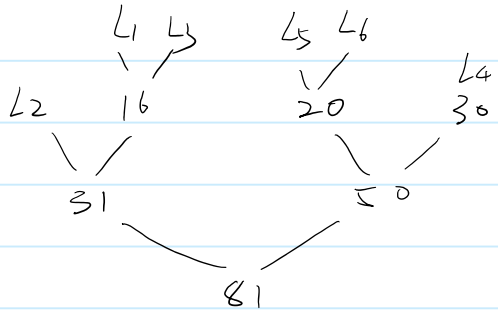


Preorder Before Next

Preorder

Inorder

L_1	L_2	L_3	L_4	L_5	L_6
8	15	8	30	10	10



For each L_i, Create $(L_i.size, L_i)$

Insert k items into a heap H.

While H.Size > 1

$k_1 \leftarrow$ removeMin(H)

$k_2 \leftarrow$ removeMin(H)

$k \leftarrow$ Union(k_1.element, k_2.element)

insertItem(k, size, k)

end while.

Return only item in H

7

2017年2月22日 14:55

ORDERED DICTIONARIES.

Data: (key, element) pairs.

keys have a linear ordering

Methods: findElement(k) closestKeyBefore(k)

insertItem(k,e) closestElementBefore(k)

removeItem(k) closestKeyAfter(k)

closestElementAfter(k)

---

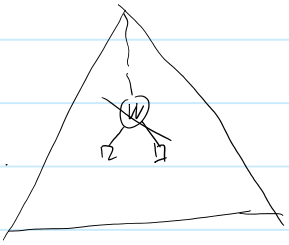
3. removeItem(k)

W ← TreeSearch(k, T, root)

CASE 1: W has two external children.

Delete W & replace it with an external node.

T is still a BST



O(1)

CASE 2: W has one external child.

Delete W & let W.parent point to the root of W's subtree.

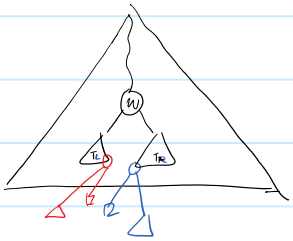
O(1)

CASE 3. W has no external child.

Delete W & replace it with the bottom rightmost node in T_L (contains max key in T_L)

or the bottom of leftmost node in T_R

(contains min key in T_R)



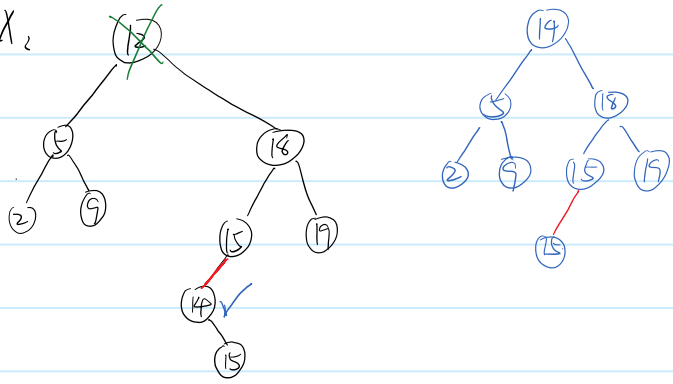
O(T.height)

Then fix problem created by node substitution using Case 1 or 2.

8

2017年2月22日 14:55

Ex:

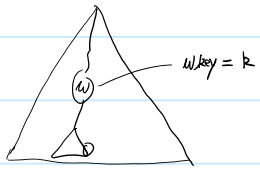


BST's

find Elements
insert Items
remove Items
} O(T.height)

closestKeyBefore(k) → TreeSearch(k, T.root)

closestKeyAfter(k)



InOrderBefore(w) O(T.height)

InOrderAfter(w)

T → n items

$\log n \le T.height \le n-1$

method run in O(n).

n items.

what determines the height of a BST ?

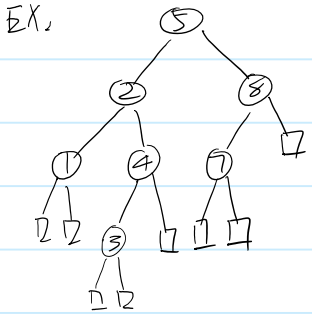
The order in which the items are inserted affects the height of BST

AVL Trees (Balanced BST)

ALL Trees are BST's with the height balanced condition.

— For every internal node v of T, the height of the left and right subtrees of v can differ by at most 1

EX.



CLAIM: Suppose T is an AVL Tree of n items

then the height is $O(\log n)$

at most $2(\log n + 1)$

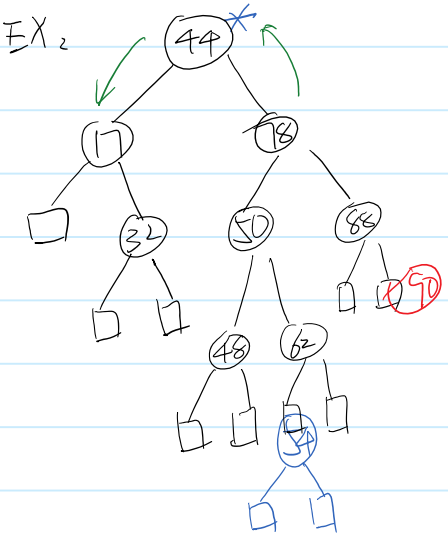
findElement(k) ⟶ Tree search $O(\log n)$

insertElement(k)

removeElement(k)

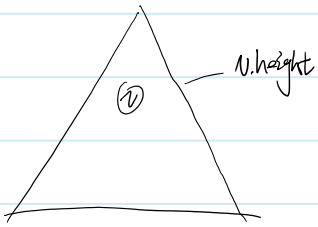
InsertItem(k,e) Insert 90, 54.

EX:



Insert Step:

AVL Tree



Insert seem like BST (i.e. find external node w for insertion of item)

Rotate w's ancestor path &
update height value when necessary
& check height balance property is violated.

Week 6

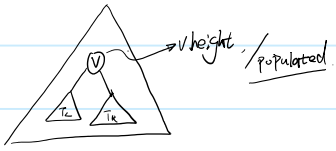
2017年2月27日 15:07

AVL TREES

— Balanced BSTs

— BST + height property $\quad |T_L.height - T_R.height| \leq 1$

O(log n)



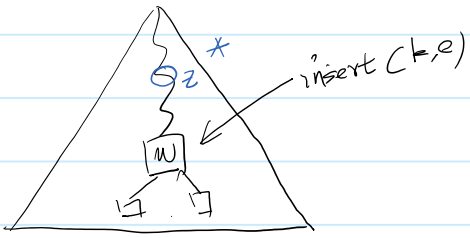
findItem(k) — like BST

insertItem(k,e) *

removeItem(k)

insertItem(k,e)

To fix violation, do Rotations

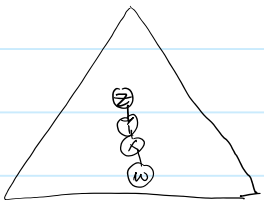


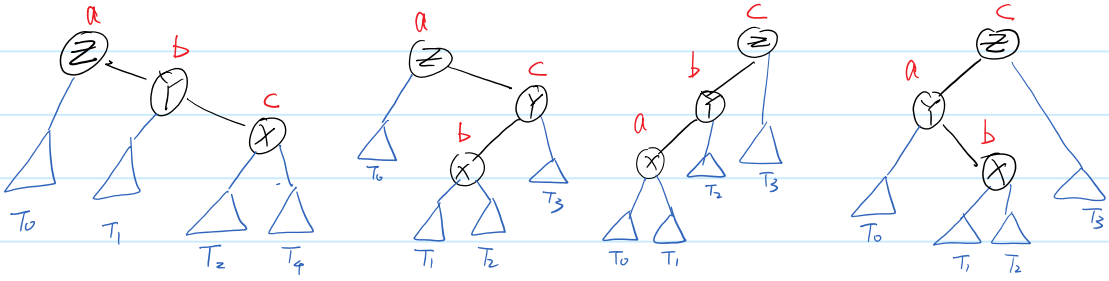
Let z = lowest ancestor of w, where an imbalance has occurred

y = child of z that is an ancestor of w

x = child of z that is an ancestor of w

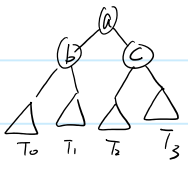
4 possible configurations





Rename, z, y, x as a,b,c so that key(a) ≤ key(b) ≤ key(c)

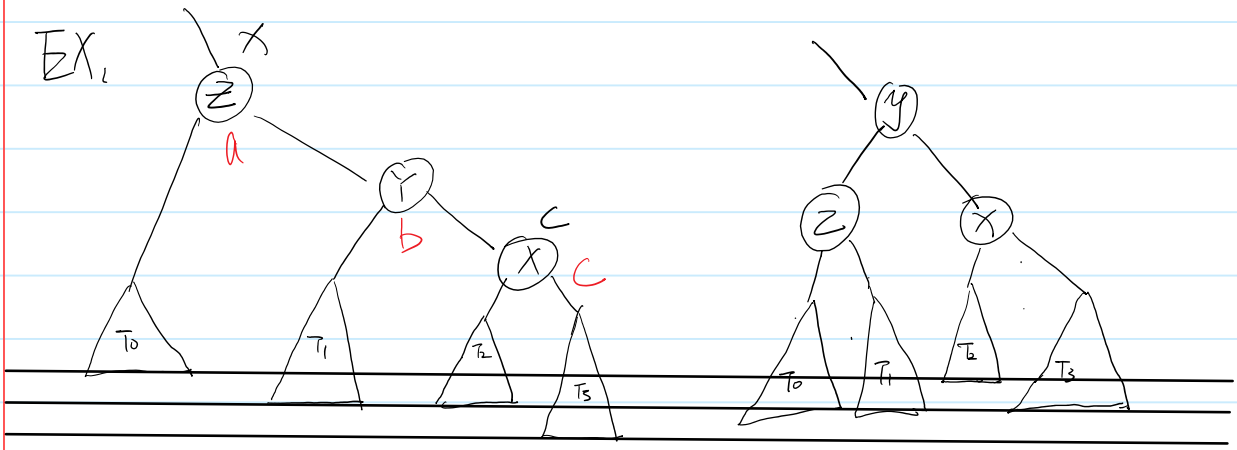
FIX: replace node z by node b. Make b's leftchild & b's rightchild be c. Then



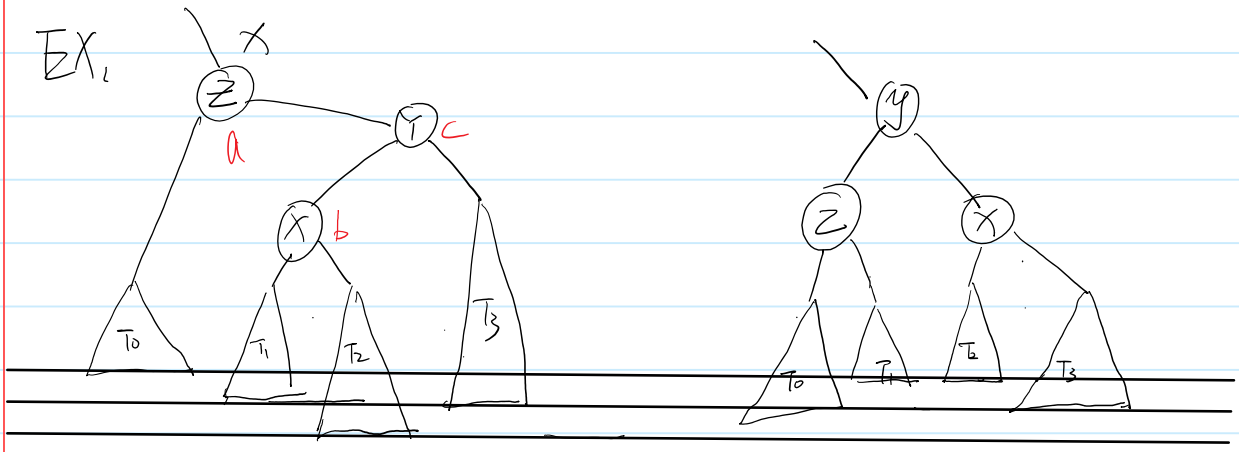
2

2017年2月27日 15:07

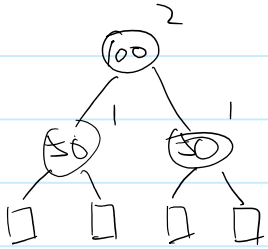
EX.



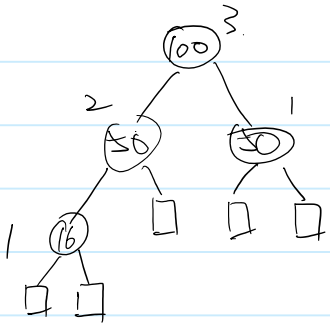
EX.

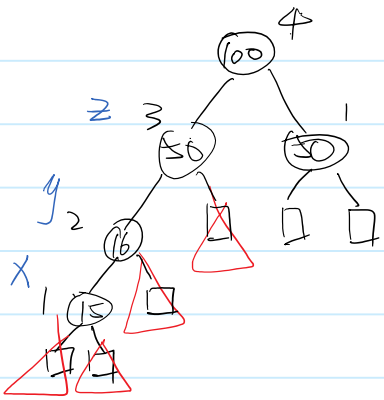


EX.



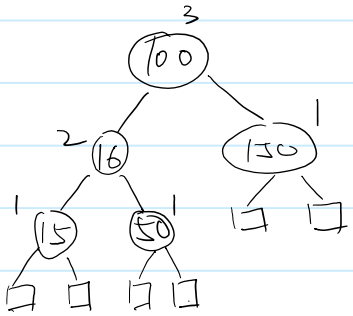
Insert 16, 15, 4

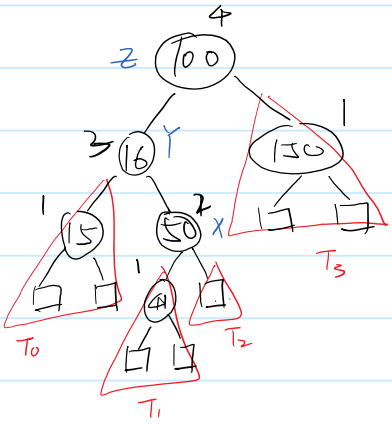




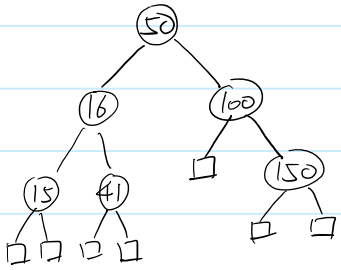
3

2017年2月27日 15:07





HW



changing pointers.

O(1) time !!!

Insert Item (k,e)

1. like a BST, find an external node w that contain (k,e) O(logn)
2. Traverse the ancestral path from w to the root to update height fields & check for imbalance. O(logn)
3. If an imbalance occurs at z, Fix problems via a Rotation O(1) update the height field of z, y, x O(1)

⇒ O(logn) time !!

Remove Item (k)

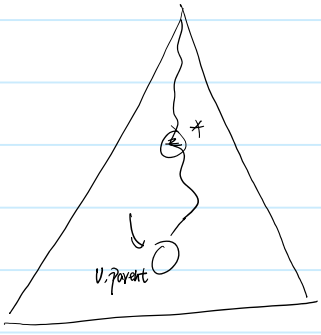
When removing an item from node w

- Replace with new item / new node like in a BST O(logn)
- Suppose the replacement was from node w. O(logn)

  Starting with w's parent up to the root,

  - update height fields
  - check for imbalances
  - Fix using rotation !!! O(logn)

    ↳ continue until the root node !!!



let z = lowest ancestor of v.parent where imbalance occured.

y = taller child of z

x = taller child of y

OR if tied, the child one the "same side as y" should chosen

⇒ O(logn)

2017年2月27日 15:07

SPLAY TREE

↳ mechanism: splaying ("move to the root operation")

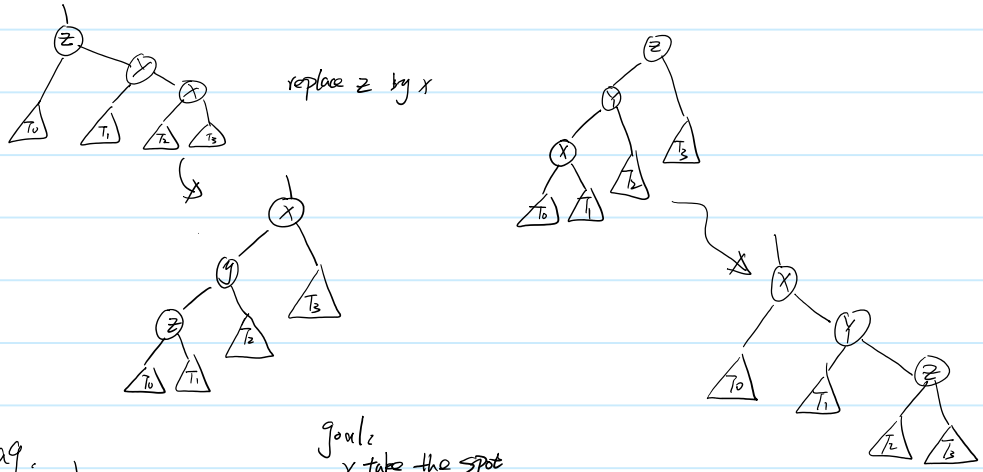
splay X —— move X to the root ...

Restructing:

Let y be the parent of X

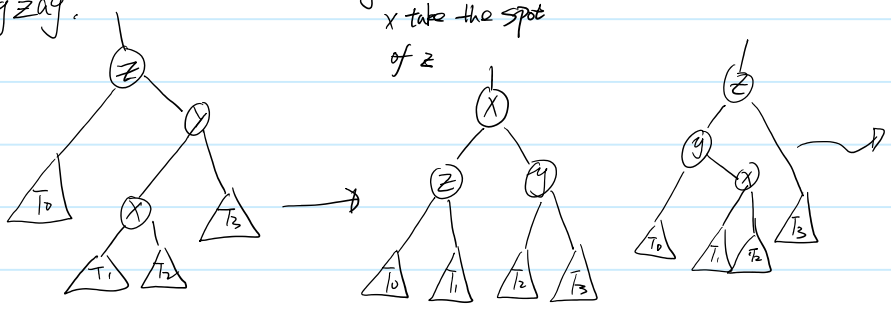
z be the grandparent of X.

Case 1. zig zig

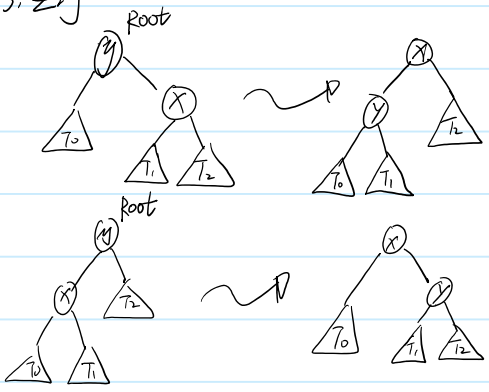


Case 2. zig zag.

goal: x take the spot of z

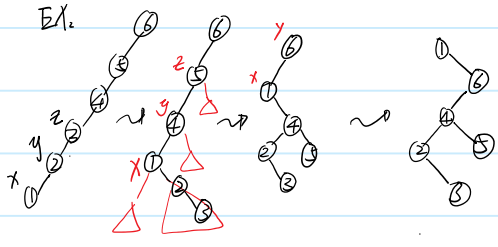


Case 3. zig



Splaying X consist of repeatedly applying the Restructing until X is the root.

Ex.

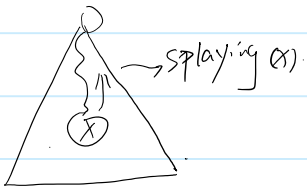


5

2017年3月1日 15:57

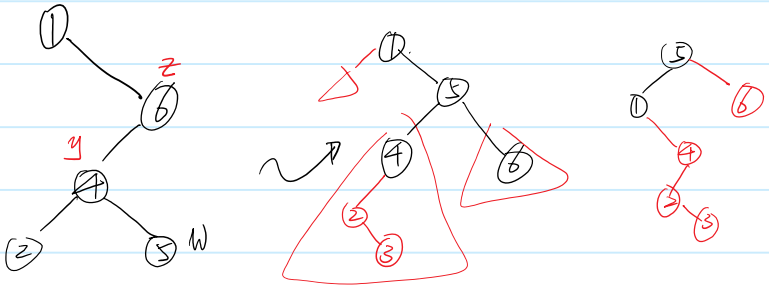
Splay Trees

– are BSTs + restructuring



Restructuring

① zig-zig ② zig-zag ③ zig



splay (w)

When to splay?

(1) FindElement (k)

w ← TreeSearch (k) O(height)

If w is an external node, splay w. O(height)

Else if w is an external node, splay the parent of w.

2. InsertItem (k,e)

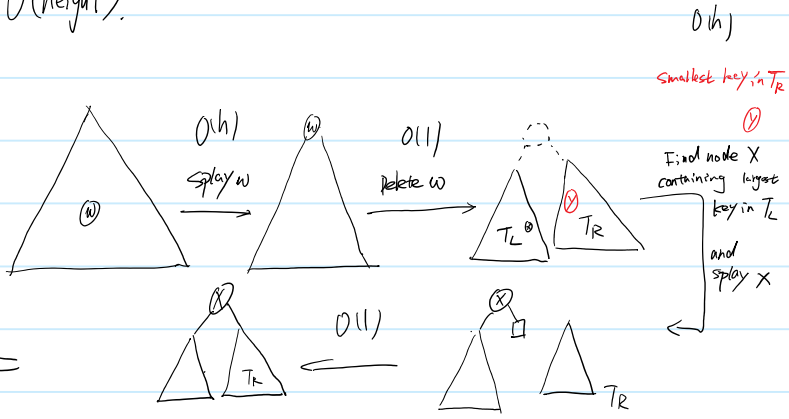
when inserting item at node w O(height)

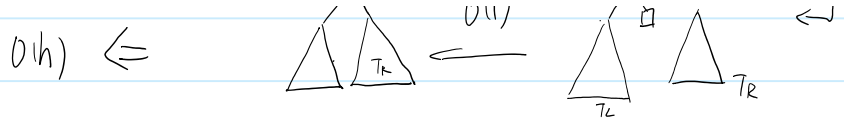
splay w.

3. removeItem (k)

(slightly different from the book)

suppose w is the node to remove





(viii) $\Leftarrow$

2017年3月1日 15:57

Advantages.

(1) Easy to implement

(2) "self-adjusting" BST

(3) Each operation takes $O(\log n)$ time on average.

THM:

Consider a sequence of n operations on a splay tree, each a search, insertion or deletion, starting from an empty splay tree. Let n_i be the # of items on the tree after operation i, and Let n = total # of insertions.

the the total running time for performing the sequence of operation is

$$O\left(m + \sum_{i=1}^{m} \log n_i\right) = O(m \log n)$$

$\hookrightarrow O(\log n)$ / operation on average.

Week7

2017年3月6日 15:58

Donald Knuth

SORTING

— HeapSORT — $O(n \log n)$

n items

① BuildHeap with n number $O(n)$

Insert n numbers into a heap $O(n \log n)$

② Do n removeMin.. $O(n \log n)$

∴ $O(n \log n)$

{ Bubble sort
| Insertion Sort $O(n^2)$
| SelectionSort

Merge Sort

IDEA: Divide and conquer

Let's S be a list of n numbers.

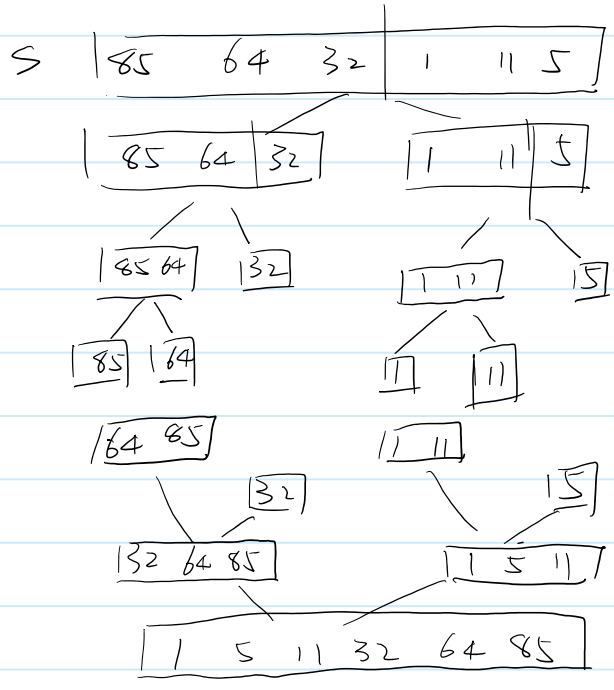
Divide: partition S into two list S_1 and S_2 of about the same size.

Conquer: Recursively sort S_1 and S_2 and merge to obtain the result.

MergeSort(S)

```
If S.size > 1
    (S1, S2) ← Partition(S)        O(S.size)
    S1 ← MergeSort(S1)
    S2 ← MergeSort(S2)
    S ← Merge(S1, S2)              O(S.size)
end if
return(S)
```

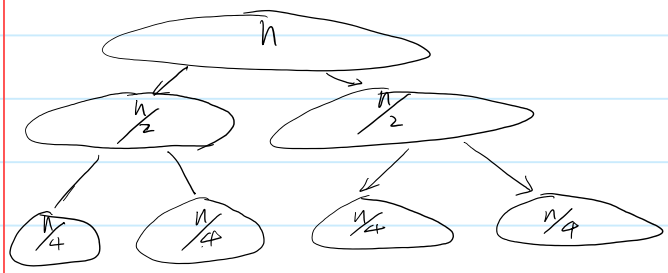
No Recursive Call $O(S.size)$



2

2017年3月6日 15:58

Running Time



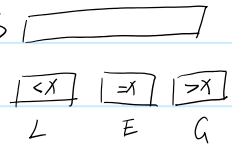
$O(n) \times O(\log n) = O(n \log n)$

QuickSort

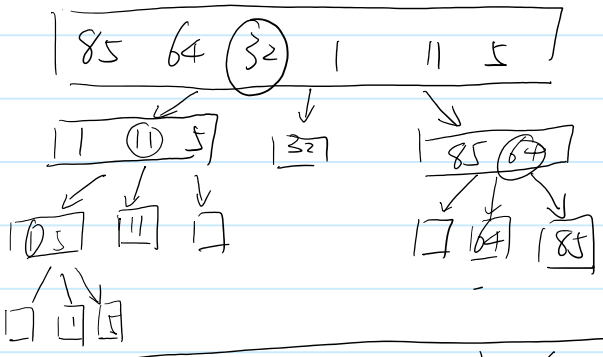
Let S be a list of n numbers.

Divide: pick a pivot element x in S.

partition S into 3 sets.



conquer: Recursively sort L & G and then merge L, E, G.



QuickSort (S)

if S.size > 1

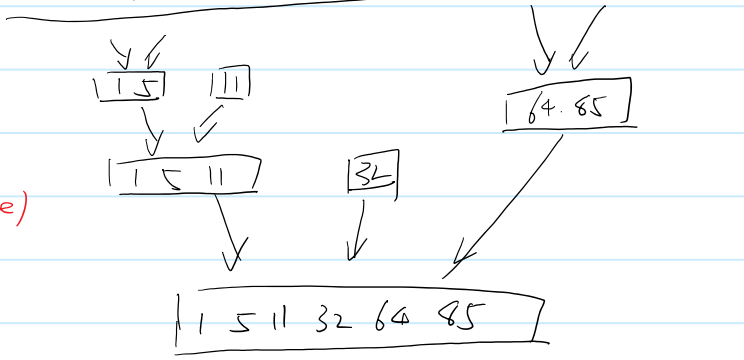
pick a pivot element x

(L, E, G) ← partition(S) O(S.size)

L ← QuickSort(L)

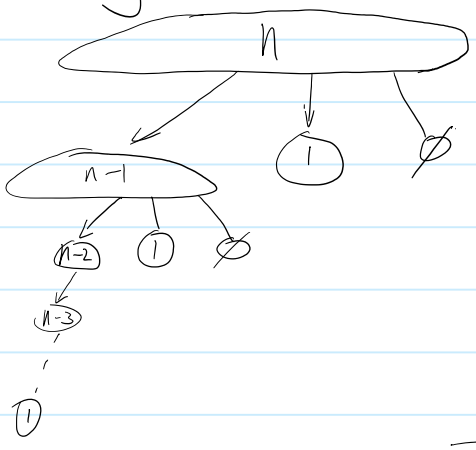
G ← QuickSort(G)

S ← merge(L, E, G). O(1)



2017年3月6日 15:57

Running Time



$O(n)$

$O(n-1)$

$O(n-2)$

$O(n-3)$

$O(1)$

$O(1+\cdots+(n-1)+n) = O(n^2)$

Randomized Quicksort

* pick a pivot uniformly at random

THM: The expected running time of Randomized Quicksort is $O(n \log n)$ on average

Proof Sketch:

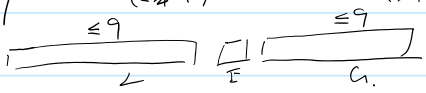
In QuickSort(S), let $m = S.size$

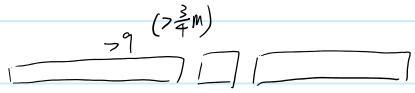
we say that a pivot x is "good" if in the partition (L, E, G) $L.size \overset{\text{and}}{=} G.size \leq \frac{3}{4}m$

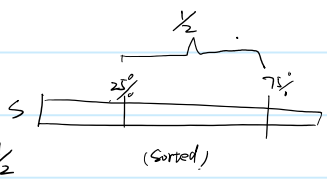
"bad" if in (L, E, G), $L.size$ or $G.size > \frac{3}{4}m$

EX: S = 1. 2. 3. 4. 5 6 7 8 9 10. 11 12

$m = 12$, $\frac{3}{4}m = 9$ $(\leq \frac{3}{4}m)$ $(\geq \frac{1}{4}m)$

good pivots 

bad pivots 



Q: if Pivot X is chosen uniformly at random, Prob(X is good pivot) = ½

2017年3月8日 15:57

Randomized Quick Sort (S)

if S.size > 1

pick a pivot element s uniformly at random.

(L, E, G) ⟵ partition(S, x)

L ⟵ RandomizedQuickSort(L)

G ⟵ RandomizedQuickSort(G)

S ⟵ merge(L, E, G)

THM: The expected running time of RandomizedQuicksort is $O(n \log n)$ time.

[Worst-case Running time $O(n^2)$]

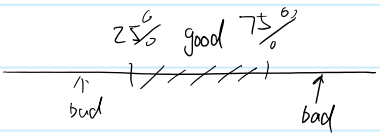
adv

* implement in place

efficient in large size.

Proof sketch:

good pivot:

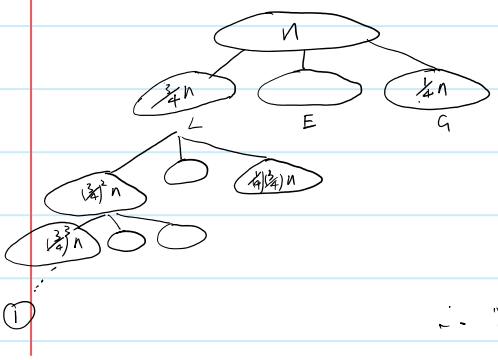


bad pivot:

$P(\text{good pivot}) = 1/2$

$P(\text{bad pivot}) = 1/2$

what if we're lucky we always choose good pivot?



$(\frac{3}{4})^h \cdot n \le 1$

$n \le (\frac{4}{3})^h$

$\log_{\frac{4}{3}} n \le h$

$h \sim \log_{\frac{4}{3}} n \approx O(\log n)$

∴ running time is $O(n \log n)$

5

2017年3月8日 15:57

A Lower Bound on Comparison-Based Sorting Algs.

HeapSort, MergeSort > $O(n \log n)$

RanQuickSort —

Can't be beaten. if most basic operation involves comparison ($x \overset{?}{>} y$)

THM: the running time of any comparision-based sorting algorithm is $\Omega(n \log n)$ where n is the number of elements being sorted.

Proof sketch: When you ask k yes/no questions.

you get 2^k different answers.

(or you can distinguish between 2^k objects)

If there are n element to sort, then there are $n!$ ordering to worry about.

To determine the right coding, the sorting algorithms will make k comparisons. s.t. $2^k \geq n!$

$2^k \geq n!$

$k \geqslant \log n!$ $\quad \log_2 n! = \log_2 (n(n-1)(n-2)-1) = \underbrace{\log_2 n + \log(n-1)}_{\frac{n}{2} \text{ items}} - - + \log_2 1$

$\geqslant \log \frac{n}{2} + \log \frac{n}{2} \cdots \log_2 \frac{n}{2}$ $\quad (\frac{n}{2} \text{ items})$

$k > \log_2 n! \geq \frac{n}{2} \cdot \log \frac{n}{2}$

is $\Omega(n \log n)$

$\therefore$ # of comparison is $\Omega(n \log n)$

2017年3月8日 16:04

Bucket Sort

Input: n items (k_1, e_1) (k_2, e_2) ... (k_n, e_n) (key, elements)

keys are integers in the range $[0, N-1]$

not comparison-based.

GOAL: Sort the items based on keys !!

Steps.

$O(N)$ ← 0. Create buckets $B[0], B[1], \ldots, B[N-1]$

$O(n)$ ← 1. Insert each item (k_i, e_i) into bucket $B[k_i]$

$O(n+N)$ ← 2. Let L be a empty List.

For $j = 0$ to $N-1$.

for each item in $B[j]$.

remove item from $B[j]$ and add to L

$O(n+N)$

$N = 10$

[9]

|

[2] ← 0, 0, 0.

[1] ∅

[0] ← 0, 0, 0.

Bucksort runs in $O(n+N)$

If $N = O(n)$ then.

it runs in $O(n)$ time.

Week9

2017年3月13日 16:04

Lower Bound on Comparison-Based

Starts at n items $\Omega(n \log n)$

Bucket Sort

① can only sort integer keys (n of them)

② all values n_i

$0 \le n_i < N$

running time	
$O(N)$	Step 1. Create N buckets $B[0]$ $B[1]$ -- $B[N-1]$.
$O(n)$	Step 2. for each item (k_i, e_i) insert into $B[k_i]$
$O(n+N)$	Step 3, let L be an empty list for each bucket $B[j]$, where $j \in [0, N)$ for each item (k,e) in $B[j]$ remove (k,e) from $B[j]$ and insert into L (append)

Bucket Sort runtime:
$O(n+N)$

An aside:

When the keys at the same items are equal, one of the desirable properties at sort is that its "stable"

DEF: An (sorting) algorithm is stable iff for every two items (k_i, e_i) and (k_j, e_j), whenever $k_i = k_j$, and $i < j$, then (k_i, e_i) precedes (k_j, e_j) in the output.

EX: $(2, e_1)$, $(1, e_2)$, $(1, e_3)$, $(1, e_4)$ ⇐ Input

$(1, e_1)$, $(1, e_3)$, $(1, e_2)$, $(2, e_1)$ not stable!

$(1, e_1)$, $(1, e_3)$, $(1, e_4)$, $(2, e_1)$ stable } output

Bucket Sort can be made stable if the bucket is queues

FIFO

Bradix Sort

lexicographically

2

2017年3月29日 14:00

Bucket Sort

— Sorting integer keys in $[0, N-1]$

— n keys $\Rightarrow O(n+N)$

Stable Sort

$(k_1, e_1)\ (k_2, e_2)\ \cdots\ (k_n, e_n)$

If $k_i = k_j$, then output should have (k_i, e_i) before (k_j, e_j) where $i<j$.

RADIX SORT

Input: ordered pairs of keys: $(k_1, l_1)\ (k_2, l_2)\ \cdots\ (k_n, l_n)$

GOAL: sorted these ordered pairs.

LD: lexicographic ordering

i.e. $(k_i, l_j) < (k_j, l_j)$ if:

(i) $k_i < k_j$

(ii) $k_i = k_j$ but $l_i < l_j$

(5,9)
(1,8)
(3,2)

(1,8) (3,2) (5,9)

$(k_1, l_1)\ (k_2, l_2)\ \cdots\ (k_n, l_n)$

Step 1: Sort the ordered pairs by second component.

Step 2: Stably sort ordered pairs by first component.

Ex: (8,3) (4,5) (7,7) (4,6) (9,5) (3,2)

Step 1 (3,2) (8,3) (4,5) (9,5) (4,6) (7,7)

Step 2 (3,2) (4,5) (4,6) (7,7) (8,3) (9,5)

3

2017年3月29日 14:00

Proof of correctness.

Consider two ordered pairs (k_i, l_i) & (k_j, l_j)

s.t. $(k_i, l_i) < (k_j, l_j)$

show Algorithm will output (k_i, l_i) first before (k_j, l_j).

CASE 1: $k_i < k_j$

step1. Sort according to l_i & l_j.

step2. Since $k_i < k_j$,
algorithm will output (k_i, k_j)
before (k_j, l_j)

CASE 2. $k_i = k_j$ & $l_i < l_j$.

→ Step 1: Since $l_i < l_j$.
(k_i, l_i) will be ahead of (k_i, l_i)

— step 2: since $k_i = k_j$ & Sorting is stable, (k_i, l_i)

* Assumption: ALL k_i, l_i's are integers and in the range $[0, N-1]$

→ Sorting using stable bucket sort.

→ Running Time: $O(n+N)$

4

2017年3月29日 14:00

SELECTION.

Given: n numbers $x_1, x_2 \cdots x_n$

GOAL: Find the k^{th} smallest number.

Sort n numbers $O(n \log n)$

Find k^{th} smallest $O(n)$

Randomized QuickSelect (S, k)

if S.size > 1

  Pick a random pivot x from S

  (L, E, G) ← partition (S, x) $O(S.size)$

$<x$	$=x$	$>x$
L	E	G

  if L.size ≥ k

    return (QuickSelect (L, k))

  else

    if L.size + E.size ≥ k

      return x

    else

      return (QuickSelect (G, k − L.size − E.size))

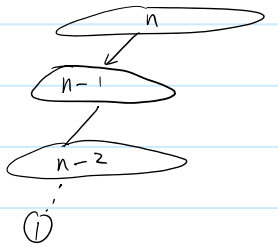
EX: 85, 24, 63, 45, 17, 31, 96, 50 k=4

24, 17, 31	45	85, 63, 96, 50
L	E	G

✓

Running Time:

Worst case



$O(n)$

$O(n-1)$

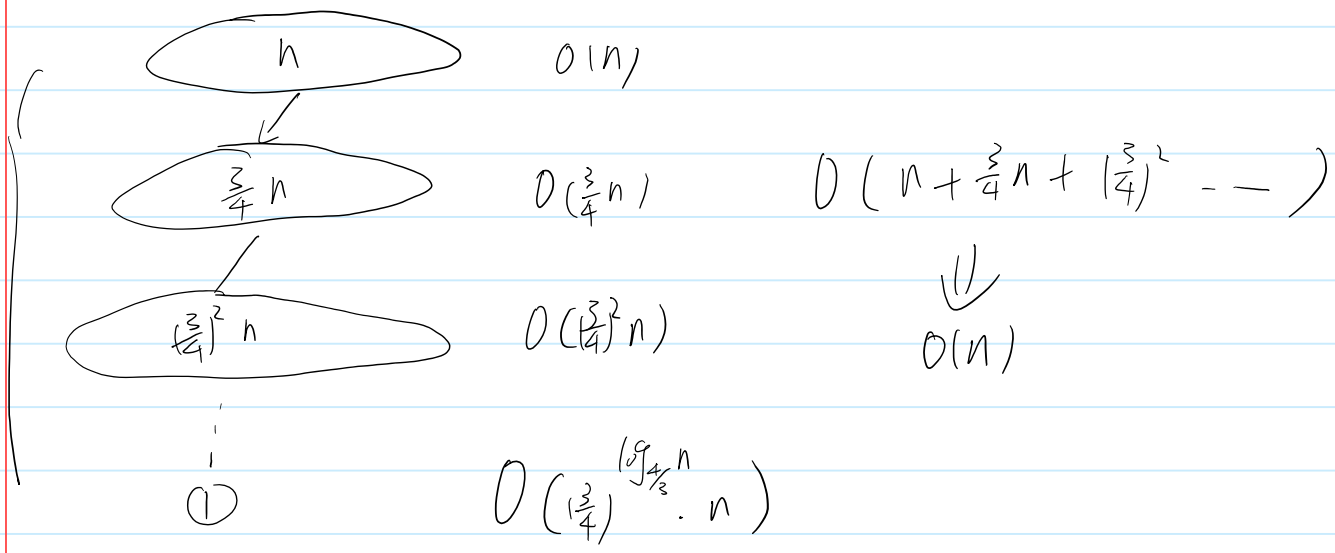
$O(n-2)$

$O(1)$

Total: $O(n + n-1 \cdots 1) = O(n^2)$

2017年3月29日 14:01

Good pivot Vs. bad pivot.

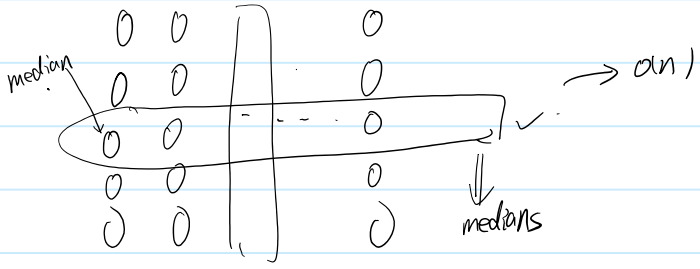


Expected running time is O(n)

Derandomization

S contains a set of numbers

pick a median of medians



Week 10

2017年4月3日 13:00

Chapter 5.

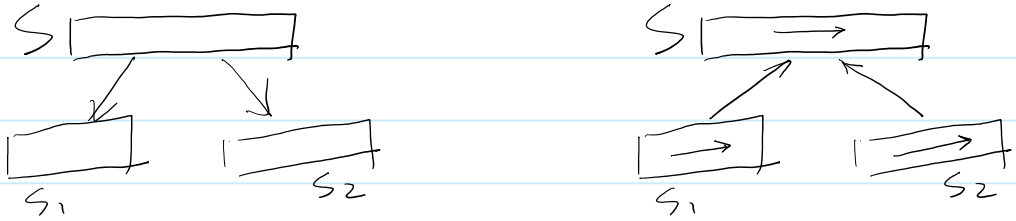
- Greedy method.
- Divide and conquer
- Dynamic programming

Divide and conquer

To solve a problem P

divide P into smaller problem that are like P.
- recursively solve smaller problems
- merge the solutions of smaller problems to form the solution of P

Classic example: Merege Sort



Runtime $O(n \log n)$ → 2 subproblems of size $\frac{S.size}{2}$ each.

→ A can to S takes $O(S.size)$ time ["non-recursive part"]

2017年4月3日 13:00

Example: Finding a celebrity

There are n people in the room.

Goal: Find a celebrity exsits.

DEF: A celebrity is a person P.
So that P doesn't know anyone in the room
but everyone else knows P

To determine if a celebrity exsits,
we can only ask questions of this type:
"person i, do you know person j"

Brute Force. $O(n^2)$

$n(n-1)$ questions will be asked.

Can we do better?

Divide & Conquer.

Name people: 1, 2, ..., n.

$n \times n$ matrix A is the "answer matrix"

$$A[i,j] = \begin{cases} 1 & \text{if person } i \text{ knows, person } j. \\ 0 & \text{O.W.} \end{cases}$$

3

2017年4月3日 13:00

CELEBRITY (M)

```
if M.size = 1
    return the only member in M.
else
    (M1, M2) ← Partition(M)
    answer1 ← CELEBRITY(M1)
    answer2 ← CELEBRITY(M2)
    if ans1 ≠ null.
        a ← ans1.
        status ← "yes".
        for each member m in M2.
            if A[a,m]=1 OR A[m,a] = 0
                status ← "no"
        end for
        if status = "yes"
            return (a)
    end if
    if ans2 ≠ null.
        b ← ans2
        status ← "yes".
        for each member m in M1.
            if A[b,m]=1 OR A[m,b] = 0
                status ← "no"
        end for
        if status = "yes"
            return (a)
    end if
    Return (null)
```

O(M2, Size)

O(M1, Size)

Running Time:

2 subproblems of size n/2 each.

Non-recursive work is

$O(\frac{n}{2}+\frac{n}{2}) = O(n)$

$\therefore O(n \log n)$

4

2017年4月3日 13:00

EX: Large Integer Multiplication

Input: two n-bit integers I & J

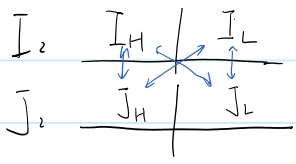
Output: I × J

Q: "usual" multiplication $O(n^2)$

```
   1101   n bits
 × 1010   n bits
```

Divide and conquer

Assume n is a power of 2 (Add 0's in front of I & J.)



$I = I_H \cdot 2^{n/2} + I_L$

$J = J_H \cdot 2^{n/2} + J_L$

I: 94/28 → I_H I_L ; $I = I_H + I_L$; $I = I_H \cdot 10^2 + I_L$

$$I \cdot J = (I_H \cdot 2^{n/2} + I_L) \times (J_H \cdot 2^{n/2} + J_L)$$

$$= 2^n \cdot I_H \cdot J_H + (I_L \cdot J_H + I_H \cdot J_L) \cdot 2^{n/2} + I_L \cdot J_L$$

```
Multiply (I, J)
  n ← # of bits of I and J.
  If n ≥ 2                                          O(n)
  {  (I_H, I_L) ← Partition (I)
     (J_H, J_L) ← Partition (J)
        a ← Multiply (I_H, J_H)
        b ← Multiply (I_H, J_L)
        c ← Multiply (I_L, J_H)
        d ← Multiply (I_L, J_L)
  }
  answer ← a × 2^n + (b+c) 2^(n/2) + d.              O(n)
  end if
  Return (I × J)
```

Running Time:

Let T(n) = worst case running time of algorithm if I & J have n bits.

$$T(n) = 4\,T(\tfrac{n}{2}) + O(n)$$

(4 = 3 ↓) → $O(n^2)$

5

2017年4月3日 13:00

Sum

$I \times J = 2^n \cdot I_H \cdot J_H + (I_L \cdot J_H + I_H \cdot J_L) 2^{\frac{n}{2}} + I_L \cdot J_L$

$(I_H - I_L)(J_H - J_L) = I_H \cdot J_H - (I_L \cdot J_H + I_H \cdot J_L) + I_L \cdot J_L$

$I_L \cdot J_H + I_H \cdot J_L = I_H \cdot J_H + I_L \cdot I_L - (I_H - I_L)(J_H - J_L)$

a d

Multiply (I, J)

n ← # of bits of I and J.

If $n \geq 2$

O(n) { $(I_H, I_L) \leftarrow$ Partition (I)

$(J_H, J_L) \leftarrow$ Partition (J)

$a \leftarrow$ Multiply (I_H, J_H)

$e \leftarrow$ Multiply $(I_H - I_L, J_H - J_L)$

$d \leftarrow$ Multiply (I_L, J_L)

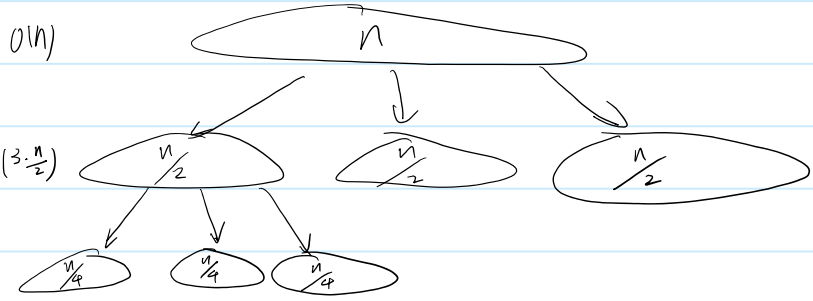
O(n) ← answer ← $a \times 2^n + (a+d-e) \times 2^{\frac{n}{2}} + d$.

end if

Return (I × J)

$T(n) = 3T(\frac{n}{2}) + O(n)$

△



$O((\frac{3}{2})^3 n)$

⋮

$O(n) + O(\frac{3}{2}n) + O(\frac{9}{4}n) - \cdots - O((\frac{3}{2})^{m} \cdot n)$

$O(n(1 + \frac{3}{2} - \cdots - (\frac{3}{2})^{\log_2 n}))$ △

$= O(n \times (\frac{3}{2})^{\log_2 n}) \approx O(n^{1.585})$

$(\frac{3}{2})^{\log_2 n} \cdot n$

$= \frac{3}{2^{\log_2 n}} \times n$

$= \frac{3^{\log_2 n}}{n} \times n$

$= n^{\log_2 3} \cdot \approx n^{1.585}$

$a^{\log_k b} = b^{\log_k a}$

2017年4月5日 16:00

LARGE INTEGER MULTIPLICATION.

n-bits: $I_H \mid I_L$, $J_H \mid J_L$

$I = I_H \times 2^{n/2} + I_L$

$J = J_H \times 2^{n/2} + J_L$

$I \times J = ?$ → $O(n^2)$

$I \times J = I_H \times J_H \times 2^n + (I_H \cdot J_L + I_L \cdot J_H) \times 2^{n/2} + I_L \cdot J_L$

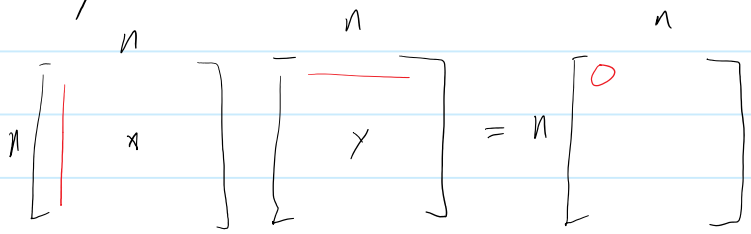
$T(n) = 4T(\frac{n}{2}) + O(n)$ → $O(n^2)$ (recursion tree)

$T(n) = 3T(\frac{n}{2}) + O(n)$

Matrix Multiplication

Input: Two $n \times n$ Matrices X & Y.

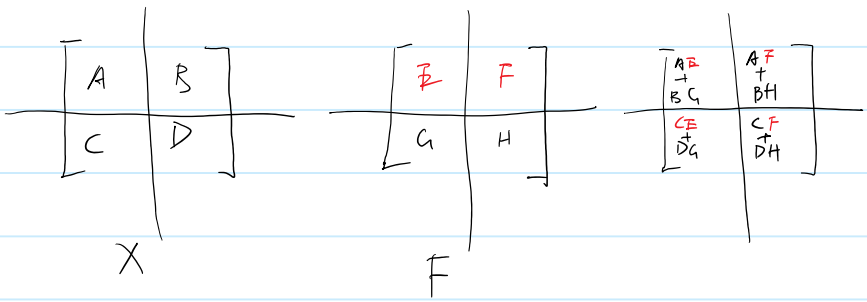
Output: $X \cdot Y$



$O(n^2)$ entries × $O(n)$ / entry $= O(n^3)$

Divide and conquer.

(Assume n is a power of 2)



$T(n) = 8T(\frac{n}{2}) + O(n^2)$

$T(n)$ is $O(n^3)$

Strassen's Alg. ~ $O(n^{\log_2 7})$ ($\log_2 7 \approx 2.808$)

(In Book)

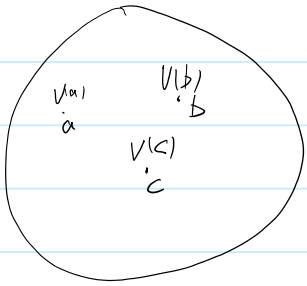
$= O(n^{-})$

7

2017年4月5日 16:01

GREEDY METHOD

- applicable mostly to optimization problems.



IDEA: Build your solution by making a sequence of decisions.

where each decision seems to be the best one when it is made.

The Set Cover Problem

Input: A set S of element $\{X_1, X_2 \cdots X_n\}$
A collection $\mathcal{S}$ of subset of S

$\mathcal{C} \leq \mathcal{S}$ covers S if every element of S is in some set of $\mathcal{C}$

Ex. $S = \{1,2,3,4,5,6,7\}$

$\mathcal{S} = \{\{1,2,3,4\}, \{1,3,5\}, \{2,4,6\}, \{5,6,7\}\}$

GOAL: Find a smallest cover for S

A cover for S: $\{\{1,2,3,4\}, \{3,5,6,7\}\}$

$\{\{1,3,5\}, \{2,4,6\}, \{3,5,6,7\}\}$

2017年4月5日 16:01

Greedy Set Cover $(S, \mathcal{S})$

$U \leftarrow S$

$b \leftarrow \emptyset$

While $U \neq \emptyset$

  Pick $T \in \mathcal{S}$ s.t. T covers the number of elements in U.

    Add T to b

    $U \leftarrow U - T$

end while

Return (b)

$S = \{1, 2, 3, 4, 5, 6, 7\}$

$\mathcal{S} = \{\{1, 2, 3, 4\}, \{3, 4, 6\}, \{1, 2, 5\}, \{7\}\}$

GREEDY Soln $= \underline{\log n} \times OPT$

$\approx$

(NP compete problem)

2017年4月5日 16:00

A scheduling problem

Input: A set of n meeting times specified by their start and finish times.

$(s_1, f_1), (s_2, f_2) \ldots (s_n, f_n)$

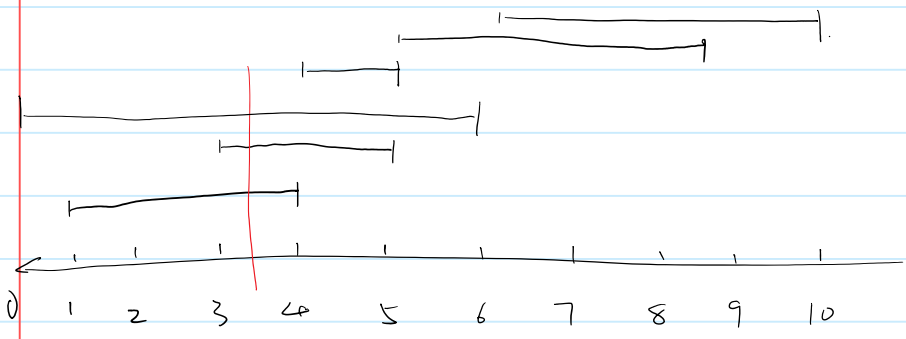
Each meeting has to be scheduled in a room.

Each room can only accomodate one meeting at a time.

GOAL: Schedule n meetings in as few rooms as possible.

Ex:

i	1	2	3	4	5	6
s_i	1	3	0	4	5	6
f_i	4	5	6	5	9	10



Greedy Scheduling $((s_1, f_1), (s_2, f_2) \ldots, (s_n, f_n))$

Sort the meeting times based on start times s.t. $O(n \log n)$

$s_1 \leq s_2 \ldots \leq s_n$

$m \leftarrow 0$ $O(1)$

For $i = 1$ to n

  If there is a room j that can accomodate meeting i. $O(n^2)$

    assign meeting i to room j

  else

    $m \leftarrow m+1$

    assign meeting i to room m.

End for

$O(n^2)$

$O(n \log n)$ Use Heap

10

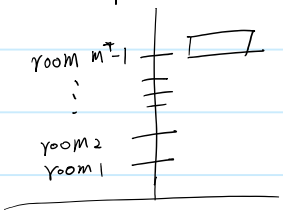
2017年4月5日 17:11

CLAIM: Greedy Scheduling will use the fewest number of rooms.

Proof: Let m^* = # of rooms used by the algorithm.

Why m^*

open room m^*

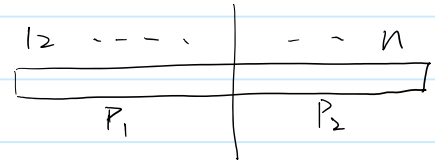


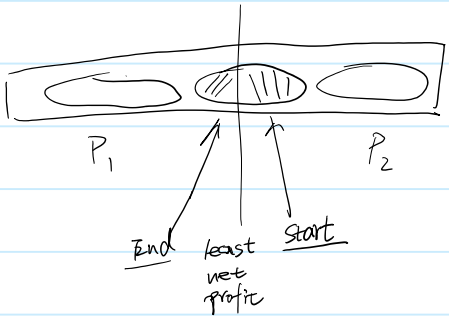
(i.e., there is m^* meeting that pairwise overlap at time t. (Lower Bound Agument)

$\Rightarrow$ At least m^* rooms had to be opened.

P_1 $M_1 > M_2$ X only $M_1 == M_2$ ✓

P_2





Compare which one is the least.

Week11

2017年4月10日 15:11

Greedy Methods

The knacksack problem.

Input: A set of n items, where item i has a benefit of b_i and a weight of w_i
A knacksack whose weight capacity is W.

Goal: Find a subset of items whose total weight is at most W and whose total benefit as large as possible.

Resource allocation problem.

0/1 knapsack problem (X) Greedy method doesn't work for 0/1
— item is in or out of the knapsack.

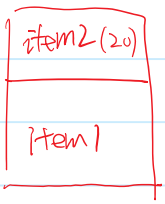
Fractional knapsack problem ✓
— allowed to use "fractions" of an item.

consider b_i/w_i ratio of all items.
Add items starting with the item with the largest ratio

EX:

i	1	2	3
b_i	200	400	240
w_i	40	80	60
b_i/w_i	5	5	4

$W = 60$



$\frac{20}{80} \times 400 = 100$
$+$
200
$= 300$

2017年4月10日 15:11

Greedy Frac knapsack $((b_1,w_1),(b_2,w_2),\ldots,(b_n,w_n),W)$

Sort the items based on their b_i/w_i ratios. —— $O(n\log n)$

Assume $\frac{b_1}{w_1} \geq \frac{b_2}{w_2} \geq \cdots \geq \frac{b_n}{w_n}$.

$i \leftarrow 1$ $w \leftarrow 0$

for $i = 1$ to n

| $x_i \leftarrow 0$ $O(n)$

end for

While $w < W$ and $i \leq n$ } $O(n)$

| $x_i \leftarrow \min\{w_i, W-w\}$

| $w \leftarrow w + x_i$

| $i \leftarrow i+1$

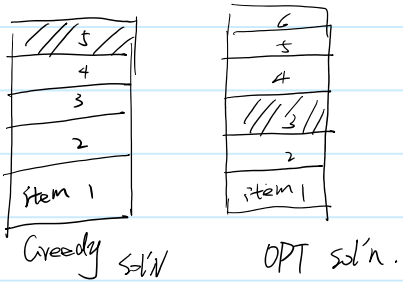
end while

Return $(x_1, x_2, \cdots, x_n)$

∴ running time $O(n\log n)$

CLAIM: Greedy knapsack chooses a set of items that maximize the total benefit.

Pf sketch:



2017年4月10日 15:11

Dynamic Programming

Recursions

— Counting problems.

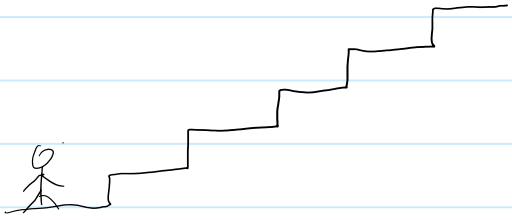
EX:

Suppose there is a stair case with n steps
You are allowed to go up 1 or 2 steps at a times.

Q.

How many different ways can you go from the bottom to the top of the staircase.

EX: n=5



1, 1, 1, 1, 1

1 2 1 1

Let S(n) = different ways a staircase with n steps can be climbed.

recursive $S(n) = S(n-1) + S(n-2)$

$= 1$ $= 2$

$= 1$ $= 2$

S3 S4

1, 2, 3, 5, 8, 13 ...

Fibonacci sequence.

Base case

$S(1) = 1$

$S(2) = 2$.

2017年4月10日 15:11

How do you compute S(n)?

Method 1: "Bottom-up" Approach ☆

Staircase(n)

Create an array S[1...n]
S[1] ← 1
S[2] ← 2
for i = 3 to n
  S[i] ← S[i-1] + S[i-2]
end for
return (S[n])

O(n)

Method 2

"Top down Approach"

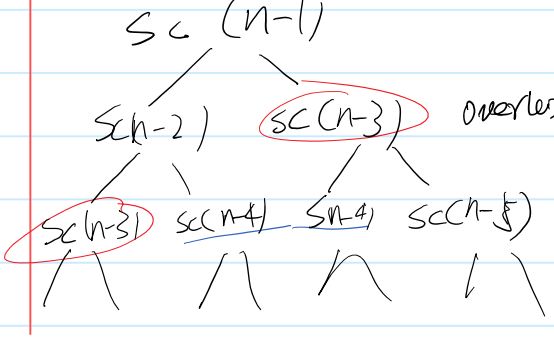
Fix: Memorization
Creat a table

Staircase 2(n)

if n=1
  Return 1.
if n = 2
  Return 2
Return (Staircase2(n-1) + Staircase2(n-2)).

Exponential running time.

overlap



(divide and conquer recursion never overlaps)

5

2017年4月12日 15:11

Dynamic Programming

— Optimization Problems.

Given a problem P

(i) find a recursive formula for solving optimal value.

(ii) State the base cases

(iii) Solve for the optimal value using bottom-up approach.

P — finding shortest path from NYC to Chicago

cost(path) = length of path

EX 1.

Week	1	2	3	...	n
LS	l_1	l_2	l_3	..	l_n
HS	h_1	h_2	h_3	..	h_n

i	1	2	3	4
l_i	10	1	10	10
h_i	5	50	5	1
	—	HL	LS	LS

max. total revenue.

total revenue = 70

i	0	1	2	3	4
OPT[i]	0	10	50	60	70

Work[i] — LS HS LS LS

Retrace

Schedule — — HS LS LS

6

2017年4月12日 16:20

Let OPT[i] = value of the optimal plan if we consider only weeks 1 to i, for i = 1 to n.

GOAL: Solve for OPT[n]

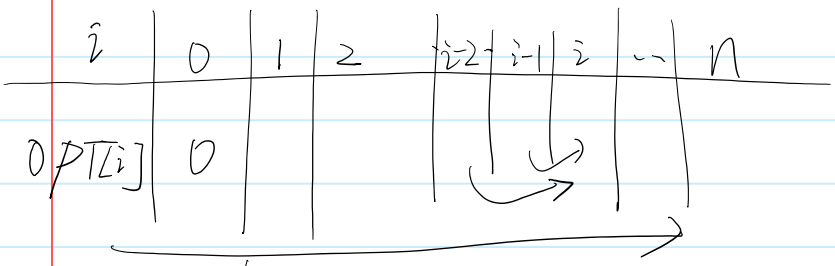
$$OPT[n] = \max \begin{cases} h_n + OPT[n-2] \\ l_n + OPT[n-1] \end{cases}$$

$$OPT[i] = \max \{ h_i + OPT[i-2], l_n + OPT[n-1] \} \quad i = 2, \dots, n.$$

Base case

$OPT[0] = 0$

$OPT[1] = \max\{l_1, h_1\}$



Size → running time / entry O(1)

O(n)

Running time O(n).

FindOPTValue($l_1, l_2, \dots l_n, h_1, h_2, \dots h_n$)

Create an array OPT[0...n]

OPT[0] ← 0

OPT[1] ← max{l_1, h_1}

for i = 2 to n

  OPT[i] = max{h_i + OPT[i-2], l_i + OPT[i-1]}

End for

Return (OPT[n])

7

2017年4月12日 16:20

EX 2

i	1	2	3	...	n
	N_1	N_2	N_3		N_n
	S_1	S_2	S_3		S_n

i	1	2	3	4
N_i	1	3	20	30
S_i	1	20	2	4

Moving cost.

Total cost = 1+3+2+4=20

Let OPT[i] = value of the optimal plan from week, to i.

$$OPT[n] = \min \begin{cases} N_n + OPT[n-1] + ?m \\ S_n + OPT[n-1] + ?m \end{cases}$$

OPT[i, NY]

OPT[i, SF]

*

2017年4月10日 15:11

#2.

a. Brute force $O(n^3)$ or $O(n^2)$

b. Least profit front Interval (i^*)
Least profit End Interval (j^*)

c. Least profit Interval [P(i ... j)]

If i=j
| Return (P[i], [i,i])
else

(P[i ... k], P[k+1 ... j]) ← Partition (P[i ... j])

(ans1, [i1, j1]) ← Least profit Interval [P(i ... k)]

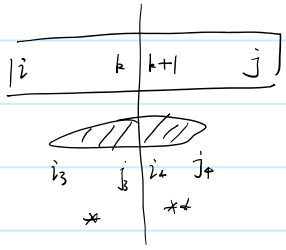
(ans2, [i2, j2]) ← Least profit Interval [P(k+1 ... j)]

O(|P.size|){

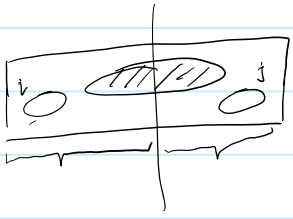
(ans3, [i3, j3]) ← Least profit End Interval (P[i ... k], k)

(ans4, [i4, j4]) ← Least profit Front Interval (P[k+1 ... j, k+1])

middle ans ← ans3 + ans4.



If ans1 = min(ans1, ans2, middle ans)
| Return (ans1,
else
if ans2 = min(ans1, ans2, middle ans)
| Return (ans2, [i2, j2])
else
Return (middle ans, [i3, j3])



O(n log n)

Dynamic programming O(n)

Week12

2017年4月17日 16:20

Solving 0-1 knapsack problem

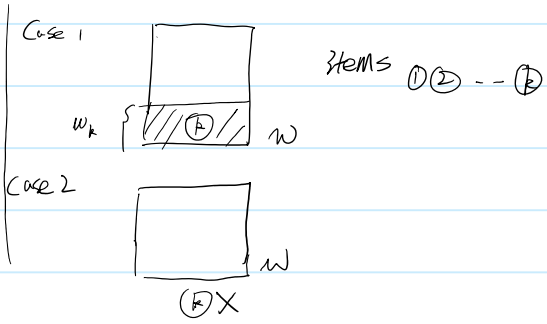
Input : n items : item $1, 2, \cdots, n$.
where item i has benefit b_i and weight w_i.
a knapsack has weight capacity W $\quad w = 1, 2, \cdots W$

Goal : choose a subset of the item s.t.
(i) total weight $\leq W$
(ii) total benefit is as large as possible.

Assume
all w_i's & W are positive integers : as indices

Define $B[k, w]$ = max benefit that can be obtained from a subset of items from $1, 2, \cdots k$.
using a knapsack of capacity w.

Solve : $B[n, W]$



recursive formula : $B[k,w] = \begin{cases} \max\{b_k + B[k-1, w-w_k],\ B[k-1, w]\} & \text{if } w_k \leq w \\ B[k-1, w] & \text{if } w_k > w \end{cases}$

$B[k, w]$

k \ w	0	1	2	⋯	w	⋯	W
0	0	0	0	⋯			0
1	0						
2	0						
⋮	⋮						
k					$B[k-1, w-w_k]$		$B[k-1, w]$
⋮							
n	0						✱

$B[k, w]$

fill up by row by row fashion

nested for-loop

Base case :
$B[0, w] = 0$ for $w = 0 \cdots W$
$B[k, 0] = 0$ for $k = 0 \cdots n$

running time : $O(n \cdot W)$ entries
$\times$
$O(1)$
$= O(n \cdot W)$

2

2017年4月17日 14:15

$O(n, W)$ ⇝ Pseudo Polynomial. NOT polynomial.

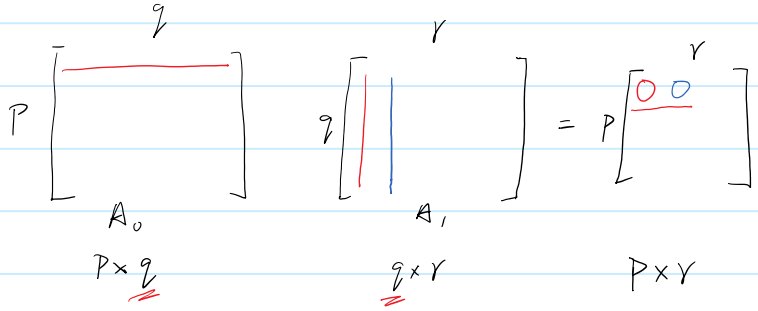
W: capacity of knapack

running time depends on W

memory size (space) ≥ $\log_2 W$ bits.

⇒ NP hard problem.

Matrix Chain Product



multiplication: $p \times q \times r$

Ex:

$A = A_0 \times A_1 \times A_2$

$= (A_0 \times A_1) \times A_2$ ①

$= A_0 \times (A_1 \times A_2)$ ②

multiplications varies by ways

A_0: 10×100 matrix

A_1: 100×5 matrix

A_2: 5×50 matrix

$(A_0 \times A_1) \times A_2$

10×5 5×50

10×50

$10 \times 100 \times 5 = 5000$ $10 \times 5 \times 50 = 2500$

$+$

$7{,}500$ multiplation

$A_1 \times A_2 \times A_0$

100×5 5×50

100×50 10×100

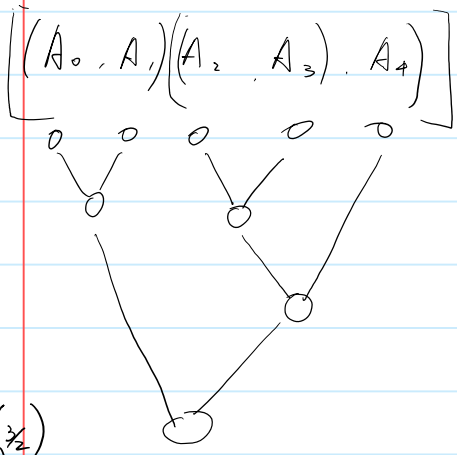
10×50

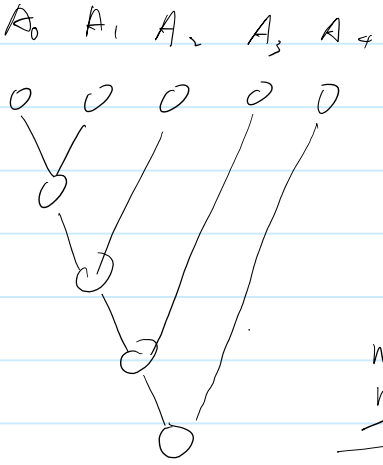
$100 \times 5 \times 50 = 25{,}000$ $10 \times 50 \times 100 = 50{,}000$

$75{,}000$

3

2017年4月17日 14:15





n^{th} Catalan number

$\Omega(4^n/n^{3/2})$

Binary tree with 5 external nodes.

n external nodes

nth Catalan Number

Exponential in n $\sim 16^n$

Given, Matrices $A_0, A_1, A_2 \cdots A_{n-1}$.

where A_i is a $d_i \times d_{i+1}$ matrix for $i = 0 \cdots n-1$

A_{i+1} is a $d_{i+1} \times d_{i+2}$ matrix

GOAL: Find a parethesization that yields the fewest # of multiplications.

Solve for $OPT[0, n-1]$

Let $OPT[i,j]$ = Minimum # of multiplications by a parenthesization of $A_i \times A_{i+1} \times \cdots A_j$

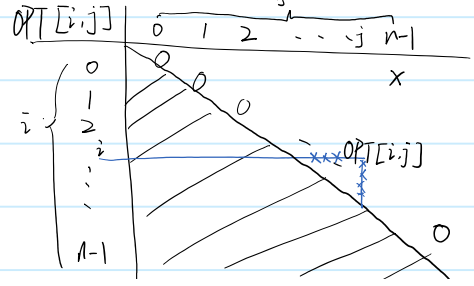
$OPT[i,k]$ $(A_i A_{i+1} \cdots A_k)(A_{k+1} \cdots A_j)$ $OPT[k+1, j]$

$d_i \times d_{k+1}$ $d_{k+1} \times d_{j+1}$

$$OPT[i,j] = \min\{ d_i \times d_{k+1} \times d_{j+1} + OPT[i,k] + OPT[k+1, j] \quad , \ i \le k \le j-1 \}$$

Base Case:

$OPT[i,i] = 0$ for $i = 0$ to $n-1$



fill table in a diagonal fashion

$A_i (A_{i+1} \cdots A_j)$

$(A_i A_{i+1})(\quad)$

$(A_i \cdots A_{k+1}) A_j$

Running Time $O(n^2) \times O(n)$ (entry, per entry)

$= O(n^3)$

fill table in a diagonal fashion

$= O(n^3)$

HW2

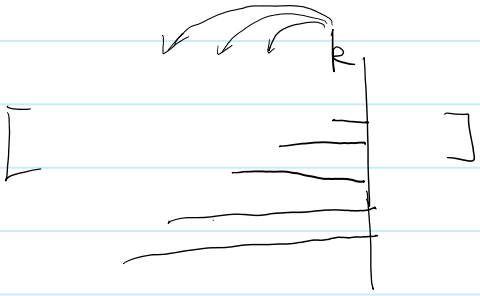
Hw9

2017年4月17日 14:15

HW9.

2. P = [1000, -250, 100, -300, 500]

WorstVal[k] = total net profit of worst interval that ends in k.



recursive formula.

O(n)

3.

Staircase google interview two-dimension.

11.2

2017年4月17日 14:15

Record

PREV[i, NY] = city that give you the smaller answer for OPT[i, NY]

PREV[i, SF] =

...low-stress jobs in weeks 3 and 4. The value of the plan is $0 + 50 + 10 + 10 = 70$.

2. Suppose you're running a consulting business. Each month, you can either run your business from an office in New York (NY) or from San Francisco (SF). In month i, you'll incur an *operating cost* of N_i if your run the bussiness out of NY; you'll incur an operating cost of S_i if you run it out of SF. However, if you run the business out of one city in month i, and then out of the other city in month $i+1$, you'll incur a fixed *moving cost* of M to switch base offices.

Given a sequence of n months, a *plan* is a sequence of n locations – each equal to either NY or SF – such that the ith location indicates the city in which you will be based on the ith month. The *cost* of a plan is the sum of the operating costs for each of the n months, plus a moving cost of M for each time you switch cities. The plan can begin in either city.

The problem. Given a value for the moving cost M, and sequences of operating costs $N_1, N_2, \ldots, N_n$ and $S_1, S_2, \ldots, S_n$, find a plan of minimum cost. (Such a plan will be called *optimal*.)

Example. Suppose $n = 4$, $M = 10$, and the operating costs are given by the following table:

	Month 1	Month 2	Month 3	Month 4
NY	1	3	20	30
SF	50	20	2	4

Then the plan of minimum cost would be the sequence of locations NY, NY, SF, SF with a total cost of $1 + 3 = 2 + 4 + 10 = 20$, where the final term 10 arises because you change locations once.

EX 2

i	1	2	3	...	n
	N_1	N_2	N_3		N_n
	S_1	S_2	S_3		S_n

moving cost: M

i	1	2	3	4
N_i	1	3	20	30
S_i	1	20	2	4

Total cost = 1+3+2+4=20

i	1	2	3	- - -	i-1	i	n
OPT[i,NY]	N_i						
OPT[i,SF]	S_i						

GOAL: Find optimal schedule. min

Let OPT[i] = value of the optimal plan from week 1 to i:

$$OPT[n] = \min \begin{cases} N_n + OPT[n-1] + ?M \\ S_n + OPT[n-1] + ?M \end{cases}$$

Let OPT[i, NY] = value of the optimal plan up to week i whose last location is NY

Let OPT[i, SF] = value of the optimal plan up to week i whose last location is SF

OPT[i, NY]
OPT[i, SF]

$$OPT[i, NY] = \min \begin{cases} N_i + OPT[i-1, NY] \\ N_i + OPT[i-1, SF] + M \end{cases} \quad \text{for } i = 2 \cdots n$$

Base case

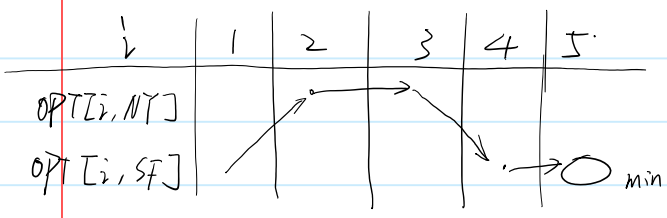
$OPT[1, NY] = N_1$

$OPT[1, SF] = S_1$

$$OPT[i, SF] = \min \begin{cases} S_i + OPT[i-1, SF] \\ S_i + OPT[i-1, NY] + M \end{cases} \quad \text{for } i = 2 \cdots n$$

11.3

2017年4月17日 16:19



Record

PREV[i, NY] = city that give you the smaller answer. for OPT[i, NY]

PREV[i, SF] =

· 2n entries

O(1) per entry

Running time, O(n)

1. [3 pts]

   a. [2 pts]

   If ith city is NY:

$$OPT(i, NY) = \min \begin{cases} OPT(i-1, NY) + N_i, prev(i, NY) = NY & \text{if (i-1)th city is NY} \\ OPT(i-1, SF) + M + N_i, prev(i, NY) = SF & \text{if (i-1)th city is SF} \end{cases}$$

   If ith city is SF:

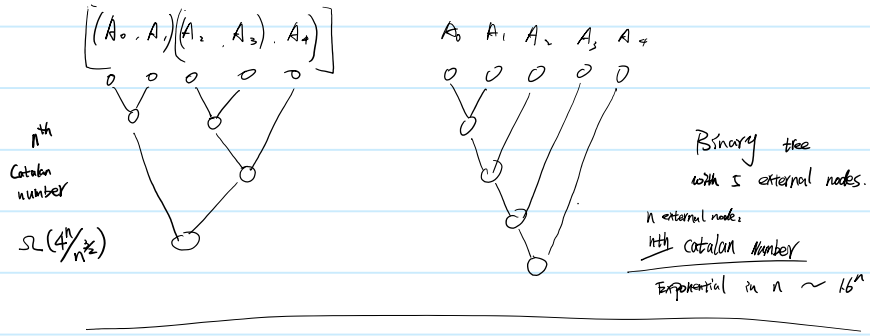
$$OPT(i, SF) = \min \begin{cases} OPT(i-1, NY) + M + S_i, prev(i, SF) = NY & \text{if (i-1)th city is NY} \\ OPT(i-1, SF) + S_i, prev(i, SF) = SF & \text{if (i-1)th city is SF} \end{cases}$$

   The optimal cost is $\min\{OPT(n, NY), OPT(n, SF)\}$.

```
OptimalPlan(N, S, n, M)
  OPT_NY[0] = OPT_SF[0] = 0
  for i = 1 to n do
    if OPT_NY[i − 1] ≤ OPT_SF[i − 1] + M then
      /* if (i-1)th city is NY and ith city is NY */
      OPT_NY[i] ← OPT_NY[i − 1] + N_i
      PREV_NY[i] ← NY
    else
      /* if (i-1)th city is SF and ith city is NY */
      OPT_NY[i] ← OPT_SF[i − 1] + M + N_i
      PREV_NY[i] ← SF
    end if
    if OPT_SF[i − 1] ≤ OPT_NY[i − 1] + M then
      /* if (i-1)th city is SF and ith city is SF */
      OPT_SF[i] ← OPT_SF[i − 1] + S_i
      PREV_SF[i] ← SF
    else
      /* if (i-1)th city is NY and ith city is SF */
      OPT_SF[i] ← OPT_NY[i − 1] + M + S_i
      PREV_SF[i] ← NY
    end if
  end for
  if OPT_NY[n] < OPT_SF[n] then
    optimalCost ← OPT_NY[n]
    optimalPlan[n] ← NY
  else
    optimalCost ← OPT_SF[n]
    optimalPlan[n] ← SF
  end if
  for i = n to 2 do
    if optimalPlan[i] = NY then
      optimalPlan[i − 1] ← PREV_NY[i]
    else if optimalPlan[i] = SF then
      optimalPlan[i − 1] ← PREV_SF[i]
    end if
  end for
  return optimalCost, optimalPlan
```

3*

Wednesday, April 19, 2017 2:15 PM



Given: Matrices $A_0, A_1, A_2 \cdots A_{n-1}$.

where A_i is a $d_i \times d_{i+1}$ matrix for $i = 0 \cdots n-1$

A_{i+1} is a $d_{i+1} \times d_{i+2}$ matrix

GOAL: Find a parenthesization that yields the fewest # of multiplications.

Solve for $OPT[0, n-1]$

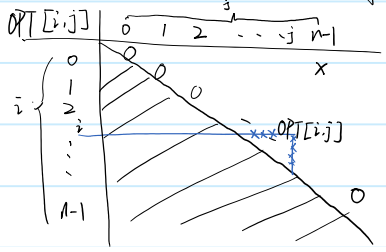
Let $OPT[i,j]$ = Minimum # of multiplications by a parenthesization of $A_i \times A_{i+1} \times \cdots A_j$

$OPT[i,k]$ $(A_i A_{i+1} \cdots A_k)(A_{k+1} \cdots A_j)$ $OPT[k+1, j]$

$d_i \times d_{k+1}$ $d_{k+1} \times d_{j+1}$

$$OPT[i,j] = \min\{ d_i \times d_{k+1} \times d_{j+1} + OPT[i,k] + OPT[k+1, j] \quad , \; i \le k \le j-1 \}$$

Base case: $OPT[i,i] = 0$ for $i = 0$ to $n-1$



fill table in a diagonal fashion

HW2

$A_i (A_{i+1} \cdots A_j)$

$(A_i A_{i+1}) (\quad)$

$(A_i \cdots A_{k-1}) A_j$

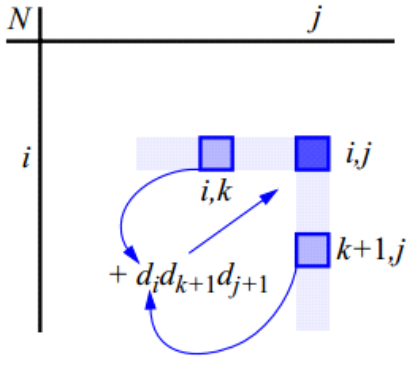
Running Time $O(n^2) \times O(n)$ (entry, per entry)

$= O(n^3)$

Algorithm MatrixChain$(d_0, \ldots, d_n)$:
Input: Sequence $d_0, \ldots, d_n$ of integers
Output: For $i, j = 0, \ldots, n-1$, the minimum number of multiplications $N_{i,j}$ needed to compute the product $A_i \cdot A_{i+1} \cdots A_j$, where A_k is a $d_k \times d_{k+1}$ matrix

for $i \leftarrow 0$ to $n-1$ **do**
  $N_{i,i} \leftarrow 0$
for $b \leftarrow 1$ to $n-1$ **do**
  for $i \leftarrow 0$ to $n-b-1$ **do**
    $j \leftarrow i+b$
    $N_{i,j} \leftarrow +\infty$
    for $k \leftarrow i$ to $j-1$ **do**
      $N_{i,j} \leftarrow \min\{N_{i,j}, N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1}\}$.

Algorithm 5.5: Dynamic programming algorithm for the matrix chain-product problem.



2017年4月19日 14:15

GRAPHS

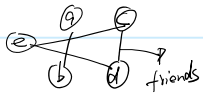
DEF: A graph is a pair (V, E) where
V is a set of vertices/nodes
E is a collection of pairs of vertices called edges

edge can be directed or undirected
- directed edge $u \to v$
- undirected edge $u - v$

Directed graph
— all edges are directed

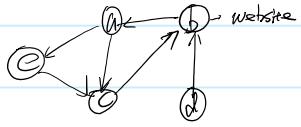
Undirected graph
— all edges are undirected

6 degrees of separation



Modeling Problems

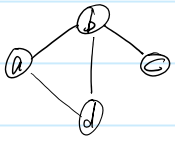
Object
— Relationship between object.
→ symmetric — Friendship Graphs
→ asymmetric — Web Graph



In graph algorithms, two parameters show up in the running time:
$n = |V|$, $m = |E|$
If input is $G = (V, E)$, the size of G is $O(n+m)$

Degrees / Indegrees / Outdegrees

Ex:



degree(a) = 2
degree(b) = 3

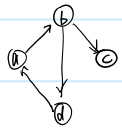
What's the min & max value of $m = |E|$?

Undirected graphs:
$0 \le m \le \frac{n(n-1)}{2} = \binom{n}{2} = (n,2)$
m is $O(n^2)$

$|V| = n$,
$0 \le \deg(v) \le n-1$

Directed graphs:
$0 \le m \le n(n-1)$
m is $O(n^2)$

Ex:



5

2017年4月19日 16:11

THM: (i) When $G \in (V, E)$ is an undirected graph, $\sum_{v \in V} \deg(v) = 2|E| = 2m$

(ii) When $G \in (V, E)$ is a directed graph.

$$\sum_{v \in V} \text{indegree}(v) = \sum_{v \in V} \text{outdegree}(v) = |E| = m.$$

$\text{Indegree}(c) = 1$

$\text{outdegree}(c) = 0$

$|V| = n$, $0 \le \text{indegree}(v) \le n-1$

$0 \le \text{outdegree}(v) \le n-1$

Sparse graph

$m = (n)$

Dense graphs

$m = \Omega(n^2)$

The Graph ADT

$O(n+m)$

$G = (V, E)$ V and E are stored in an container. usually a linked list.

Some of the information we need:

incident Edges (v)

are Adjacent (u, v)

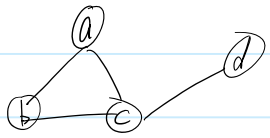
insert Vertex (v)

remove Vertex (v)

2017年4月19日 16:11

TWO Implementations.

1. Adjacency List Structure



$I(a) = \{ab, ac\}$

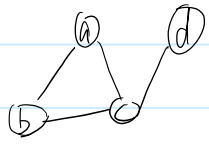
$I(b) = \{ab, bc\}$

$I(c) = \{ac, bc, cd\}$

$I(d) = \{cd\}$

SPACE: $O\left(\sum_{v \in V} \deg(v) = O(m)\right)$ ⇒ less space used in practice

2. Adjacency Matrix Structure



	a	b	c	d
a	0	1	1	0
b	1	0	1	0
c	1	1	0	1
d	0	0	1	0

too much for sparse graphs.

Space: $O(n^2)$.

	Adjacency List	Adjacency Matrix
incident Edges (v)	$O(\deg(v))$	$O(n)$
are Adjacent (u, v)	$O(\min\{\deg(u), \deg(v)\})$	$O(1)$
insert Vertex (v)	$O(1)$	$O(n)$
remove Vertex (v)	$O(\deg(v))$	$O(n)$

note

2017年4月17日 14:15

1. Suppose you're managing a team of expert computer hackers, and each week you have to choose a job for them to undertake. Now, as you can well imagine, the set of possible jobs is divided into those that are *low-stress* (e.g., setting up a website for a class at a local grade school) and those that are *high stress* (e.g., protecting the nation's most valuable secrets, or helping a desperate group of students finish a project that has something to do with compilers). The basic question, each week, is whether to take on a low-stress job or a high-stress job.

   If you select a low-stress job for your team in week i, then you get a revenue of $l_i > 0$ dollars; if you select a high-stress job, you get a revenue of $h_i > 0$ dollars. The catch, however, is that in order for the team to take on a high-stress job in week i, it's required that they do no job (of either type) in week $i-1$; they need a full week of prep time to get ready for the crushing stress level. On the other hand, it's okay for them to take a low-stress job in week i even if they have done a job (of either type) in week $i-1$.

   Given a sequence of n weeks, a *plan* is specified by a choice of "low-stress", "high-stress" or "none" for each of the n weeks, with the property that if "high-stress" is chosen for week $i > 1$, then "none" has to be chosen for week $i-1$. (It's okay to choose a high-stress job in week 1.) The *value* of the plan is determined by summing up the revenues of the n weeks.

   The problem. Given sets of values $l_1, l_2, \ldots, l_n$ and $h_1, h_2, \ldots, h_n$, find a plan of maximum value. (Such a plan will be called *optimal*.)

   Example. Suppose $n = 4$, and the values of l_i and h_i are given by the following table:

	Month 1	Month 2	Month 3	Month 4
l	10	1	10	10
h	5	50	5	1

   An optimal plan would be to choose "none" in week 1, a high-stress job in week 2, and low-stress jobs in weeks 3 and 4. The value of the plan is $0 + 50 + 10 + 10 = 70$.

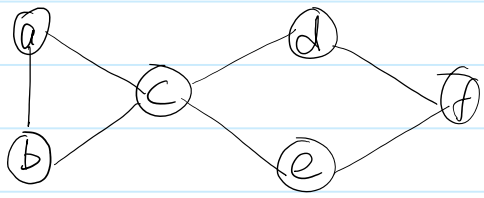
Week13

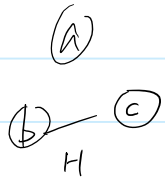
2017年4月24日 13:55

GRAPH TRAVESALS

Some terminologies:

- Walk
- Path
- Cycle





a, b, c, d, e, f, e, c is a walk of length 6
(node c, walks twice) but not a path

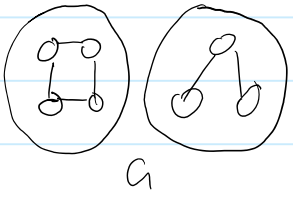
a, b, c d, f, e is a path of length 5

a, b, c, a is a cycle of length 3

Given $G = (V, E)$, a subgraph of G is another graph $H = (V', E')$ s.t.

$V' \subseteq V$, $E' \subseteq E$.

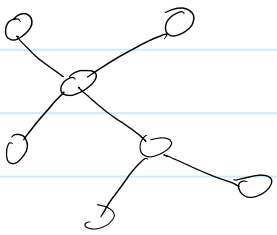
G is connected if for any two nodes u & v there is a path from u to v.

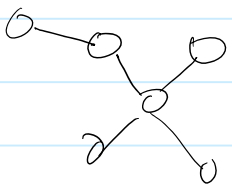


A connected component of G is a maximal connected subgraph of G.

Trees —

a connected graph with no cycles.



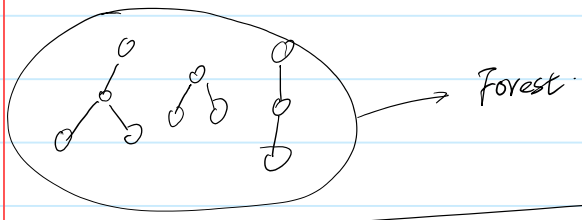


n nodes ⇝ n-1 edges

2017年4月24日 13:55

Forests

— A forest is a graph with no cycles.

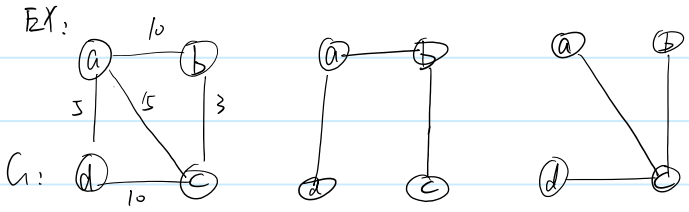


Given a graph $G = (V, E)$, $T = (V', E')$ is a spanning tree of G iff

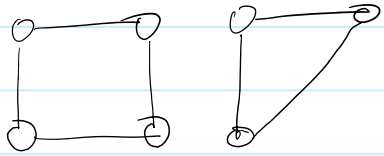
(i) T is a subgraph of G.

(ii) $V' = V$

(iii) T is a tree

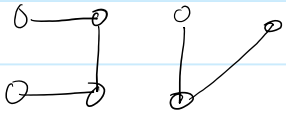


A spanning forest of G



$\subseteq G$.

Spanning forest of G.



2017年4月24日 13:55

Depth first Search (DFS)

IDEA:

Start at an arbitrary vertex v, and go as deep as you can, discovering new vertices until you can't, Back up!

Nodes: — explored or unexplored. (lables)

Edges: — unexplored, discovery, back edges.

Initially, all nodes & edges are unexpected!!

DFS (G, v):

O(1) Label v as explored.

For each edge e = vw incident to v ← adjacency List

O(deg(v)) { if e is unexplored.

O(1) { If w is unexplored.

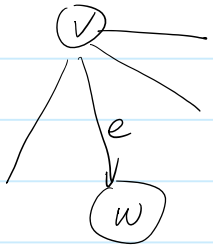
label e as a discovery edge.

pred [w] = v.

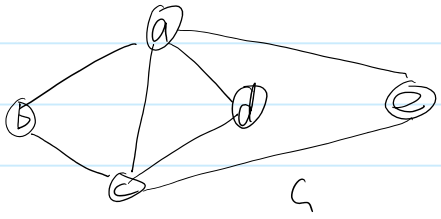
DFS [G, w)

else

label e as a backedge.



Ex:



a: ~~ab~~, ~~ac~~, ad, ~~ae~~

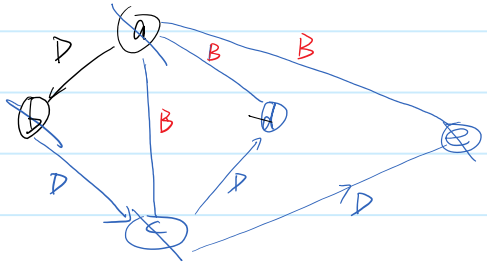
b: ~~ba~~, ~~bc~~

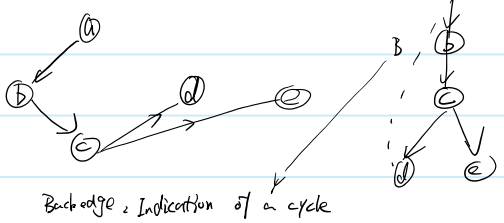
c: ca, ~~cb~~, cd, ~~ce~~.

d: ~~da~~, ~~dc~~

e: ~~ea~~, ~~ec~~.

DFS (G, a)





Backedge: Indication of a cycle

4

2017年4月24日 13:55

THM: (i) DFS(G,v) visit all vertices and edges in the connected component of v.

(ii) The discovery edges labeled by DFS(G,v) form a spanning tree of the connected component of v.

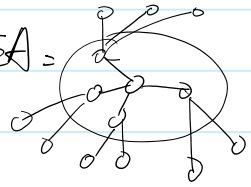
RUNNING TIME

$$O\left(\sum_{v \in V}(1+\deg(v))\right)$$

$$= O(n+m)$$

Breadth First Search (BSF)

IDEA:



Labels:

Node: (unexplored), explored

Edges: (unexplored), Discovery, Cross

BFS(G,v) (Iterator)

Create a List L_0 that contains v.

i ← 0

While L_i is not empty

  Create an empty list L_{i+1}

  For each vertex in L_i

    Label w as explored

    for each edge e=wx incident to w

      if e is unexplored.

        if x is unexplored

          add x to L_{i+1}, Label e as a discovery edge

        else

          label e as a cross edge.

    end for

  end for i ← i+1

end while

5

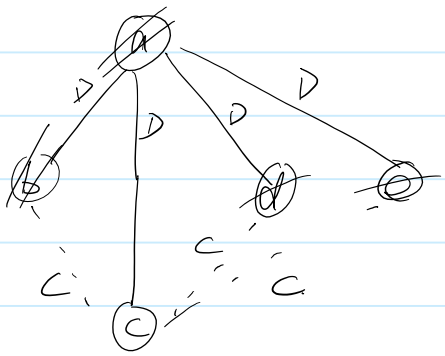
2017年4月19日 16:11

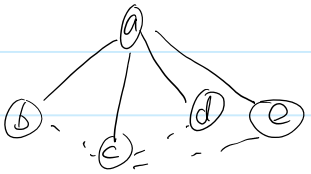
BSF (G, a)

$L_0 = \{a\}$

$L_1 = \{b, c, d, e\}$

$L_2 = \{\ \}$





DFS & BFS

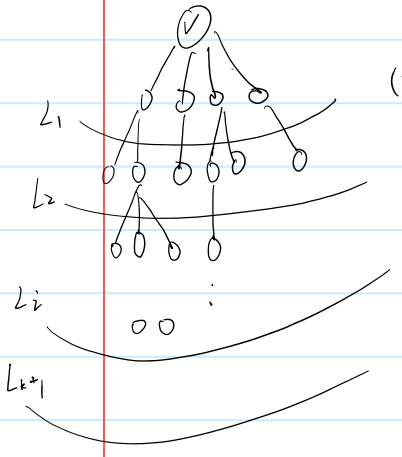
↓

$O(n+m)$

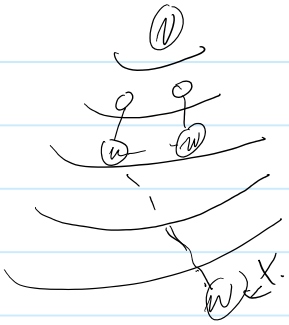
THM: (i) BFS (G, V) visits all vertices and edges in the connected component containing v.

(ii) The discovery edges labeled by BFS (G, V) form a spanning tree of the connected component of v.

(iii) For each $u \in L_i$, the shortest path from v to u contains i edges.



(iv) if uw is a cross edge, then the levels of u & w differ by at most 1.



Running Time: (Adjacency List)

$O(\sum_{v \in V} (1 + \deg(v)))$

$= O(n+m)$

Cube20.org

2017年4月26日 14:44

APPLICATIONS.

1. Testing when an undirected graph G is connected.

$O(n+m)$

$O(n)$

Run BFS (G, v) or DFS (G, v)

if all nodes were explored
| G is connected.
else
| it's not
end.

$\therefore O(n+m)$

2. For two nodes u & v of G, is there a path from u to v? If so, what's a shortest path?

Run BFS (G, u)

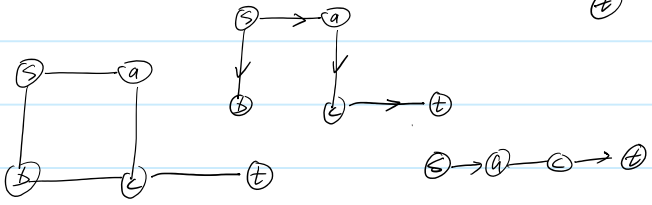
If v is explored then.
| a $u-v$ path exists.
else
| no such path exists.
end.

$O(n+m)$

pred [u]
↳ predecessor.

$s \rightsquigarrow t$

$s \dashrightarrow pred[pred[t]] \rightarrow pred[t] \rightarrow t$



2017年4月26日 14:44

3. Determine if a cycle exists in G.

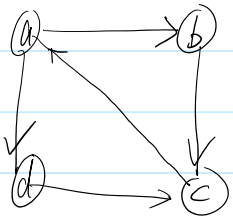
Run BFS or DFS. (on every component of G).
if a cross edge for BFS or
back edge for DFS exists.
then
G has a cycle !!

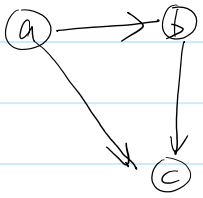
O(n+m)

Connectivity and acyclicity in directed graphs

DEF. A directed graph G=(V,E) is strongly connected if for any two nodes u & v, there is a path from u to v and from v to u.

Ex:





X

Weakly connected (If the underlying undirected graph is connected).

Q: Suppose G is a directed graph. Is G strongly connected?

For each node v of G.
Run BFS(G,v) or DFS(G,v).
If some node is unexplored.
return "no"
end For
Return "yes"

} O(n(n+m))

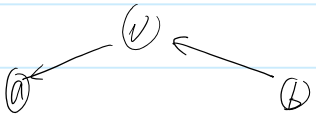
2017年4月26日 14:44

LEMMA: G is strongly connected iff for some node v.

(i) v can reach every other node in G.
(ii) every other node in G can reach v.

"⟹" "Obvious" !!

"⟸" Let a and b be any two nodes in G.



Check Strong Connectivity (G)

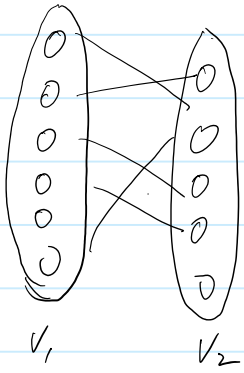
Pick a node V.
Run BFS(v) or DFS(v)
If some nodes is not explored.
Return "no"

O(m+n) Construct G^R, the graph obtained by reversing the edge of G.
Run BFS(G^R, v).
If some node is not explored.
Return "no" // some node can not reach v.
Else
Return "YE".

9

2017年4月26日 14:44

2. Bipartite graph



3.

HW #9. WorstVal[k] = net profit of worst interval that ends in k (make definition)

$$WorstVal[k] = \min \begin{cases} P[k] + WorstVal[k] \\ P[k] \end{cases}$$

WorstVal[1] = P[1]

10

2017年4月26日 14:44

Week14

2017年5月1日 14:08

$Array[X] = \max \{ Array[X - a_i] \}$, i to k, $X - a_i \geq 0$

$Array[x'] = \begin{cases} 1 \\ 0 \end{cases}$ if we can make change for x' $\quad x' = 0$ to X.

Ex: $a_1 = 3$, $a_2 = 7$

$X = 15$? $x' = 12$ if we use a 3¢

$x' = 8$ 7¢.

PossibleMakeChange $(a_1, a_2 \ldots a_k, X)$

Create an array possible $[0 \cdots X]$

Possible $[0] \leftarrow 1$

for $x' = 1$ to X.

  Possible $[x'] \leftarrow 0$

  for $i = 1$ to k

    if $x' - a_i \geq 0$ and Possible $[x' - a_i] = 1$

      Possible $[x] \leftarrow 1$

  end for

end for.

return (Possible $[X]$)

Ex: $a_1 = 3$, $a_2 = 7$, $X = 11$

x'	0	1	2	3	4	5	6	7	8	9	10	11
Possible[x']	1	0	0	1	0	0	1	1	0	1	1	0

Running time

$O(kX)$

Pseudo polynomial time !

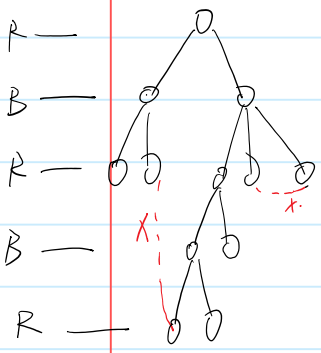
2017年5月1日 14:11

Bipartite graph.

G is bipartite. iff G has no odd cycles.

BFS-based algorithm

BFS(G, s)



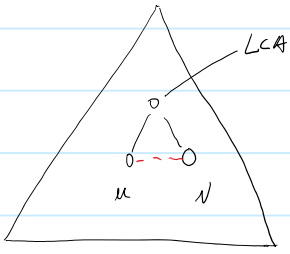
For each cross edge uv.

  if depth(u) = depth(v)

    "G is not bipartite"

endfor

"G is bipartite"



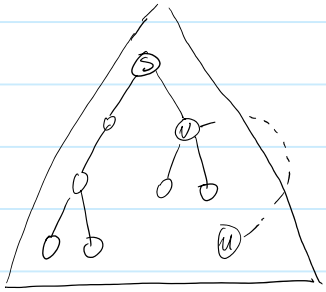
length of cycle.

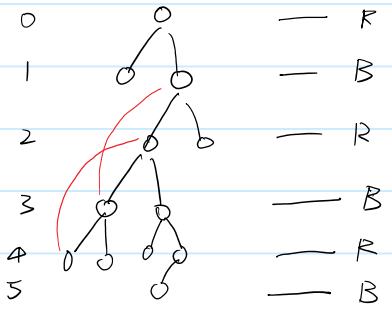
$x + x+1 = 2x+1$

odd

DFS-based algorithm

construct DFS tree





For each back edge

  if depth u & depth v are both odd or both even

    "G is not bipartite"

endfor.

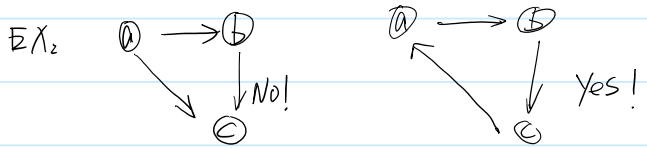
"G is bipartite".

u	v	=
even	even	even
odd	odd	even

2017年5月1日 14:11

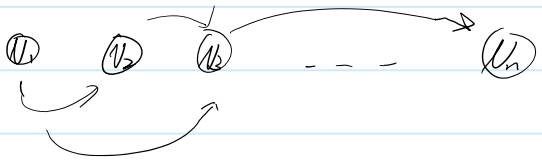
Directed graph

Given an directed G, does G contain a directed cycle?

Ex: 

For undirected graphs, run BFS / DFS & check for cross edges / back edges.

DEF: Let G be a directed graph w. n vertices.
A topological ordering/sorting of G is an ordering of the vertices of G as $(v_1, v_2, \ldots v_n)$
st. whenever v_i, v_j is an edge, $i < j$



NOTE: Graphs with topological orderings do not contain directed cycles.

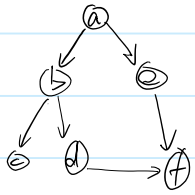
THM: G is a directed acyclic graph (DAG) iff G has a topological ordering.

Pf: "$\Leftarrow$" obvious

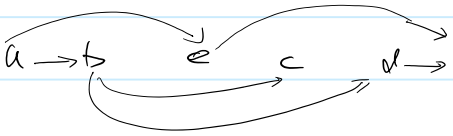
"$\Rightarrow$" Assume G is a DAG. We will describe an algorithm, that will produce a topological ordering of G.

4

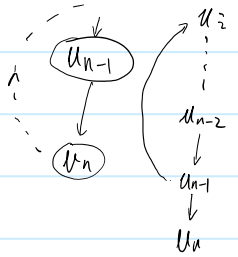
2017年5月1日 14:08



choose node with indegree 0 as v_1



LEMMA: if G is a DAG, then G has a node with indegree 0.



node on top is the node with indegree.

Observation: If G is a DAG and indegree(v) = 0 then G is still a DAG

Topological sort(G)

Store all nodes with indegree 0 in a list S.

$i \leftarrow 1$

While S is not empty

Remove a node u from S

$v_i \leftarrow u$.

$i \leftarrow i+1$

Delete v_i and all edges out of v_i. $O(\deg(v_i))$

Add node with indegree 0 to S

end while.

if $i > n$.

Return $(v_1, v_2 \dots v_n)$

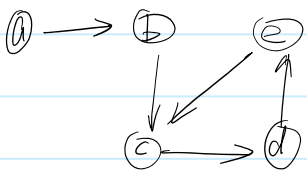
else

Return "G is not A DAG"

$O(m+n)$.

$$O\left(\sum_{v_i}(1+\deg(v_i))\right)$$

$$= O(n+m)$$

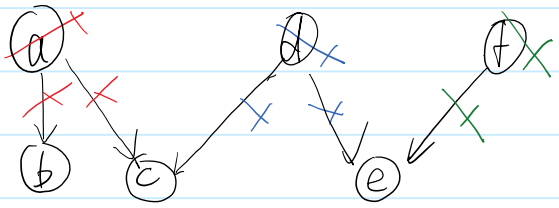


a b

v_1 v_2.

5

2017年5月3日 14:08



List

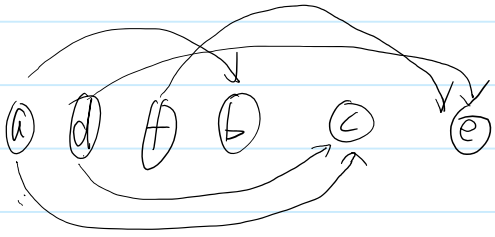
remove.

$S = \{a, d, f\}$ $v_1 = a$.

$S = \{d, f\}$

$S = \{d, f, b\}$ $v_2 = d$.

$S = \{f, b, c\}$ $v_3 = f$



$S = \{b, c, e\}$ $v_4 = b$

$S = \{c, e\}$ $v_5 = c$

$S = \{e\}$ $v_6 = e$

$S = \emptyset$

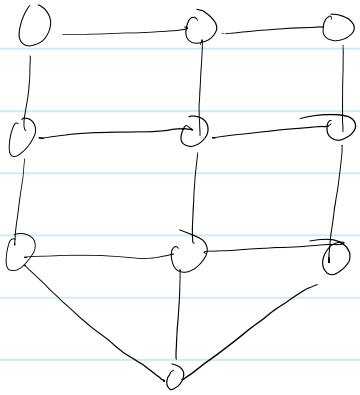
For each v, keep track of indegree[v]

Adjacency list

2017年5月3日 14:08

SHORTEST PATH

Graphs —— Weighted.



EX: Road map

Vertices — street intersections

Edges —— roads that connect street intersections.

Edge weights —— time to travel along edge

distance / length of edge, etc.

cost of traversing edge

length of path

$P = w(e_1) + w(e_2) + \cdots w(e_k)$

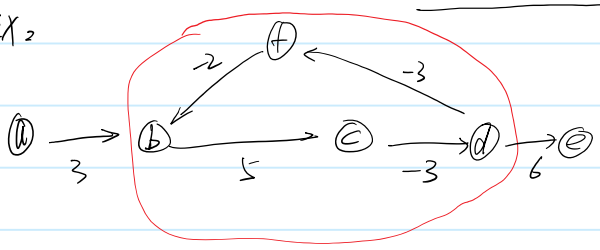
In a weighted graph (G, w),

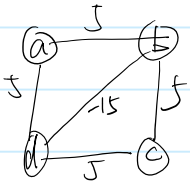
the distance from a node s to a node t is equal to the length of a shortest path from s to t. $d(s,t)$

IQ: How do we compute $d(s,t)$

* First, we note that negative-cost cycles are bad.

EX:





Problem: A shortest path distance may not be well-defined

So What do we do.

— Assume G is a DAG

— Assume edge costs in G are non-negative

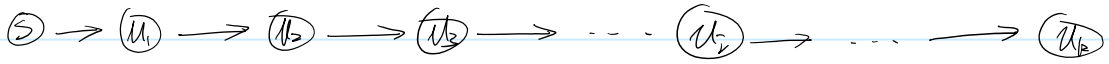
— Allow for negative-cost edges but hopes no negative-cost cycles.

7

2017年5月3日 14:08

* When solving for d(s,t) in the first two cases, we will solve the sp-distance from s to all nodes in the graph.

P_2



Suppose P is a sp from s to u_k.

Then $s, u_1, u_2, \dots u_i$ is also a sp from s to u_i.

I. Assume G is a DAG

Input: a weighted DAG, (G, w)

Starting node s

Goal: Compute shortest path distances from s to all nodes.

D[v] = sp distance from s to v.

Pred[v] = predecessor of v in a sp from s to v.

DAG-SP(G, w, s)

Compute a topological ordering of G

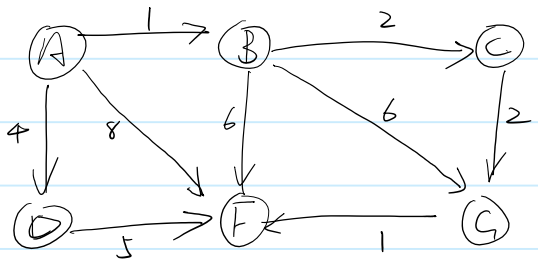
```
D[s] ← 0 , Pred[s] ← nil
For each v_i ≠ s ,                    O(n)
    D[v_i] ← +∞
    Pred[v_i] ← nil
endfor
For i = 1 to n-1                      O(n+m)
    for each edge v_i, v_j .
        if D[v_j] > D[v_i] + w(v_i, v_j)
            D[v_j] ← D[v_i] + w(v_i, v_j)
            Pred[v_j] ← v_i
    endfor
endfor
Return D-Value , Pred-Value
```

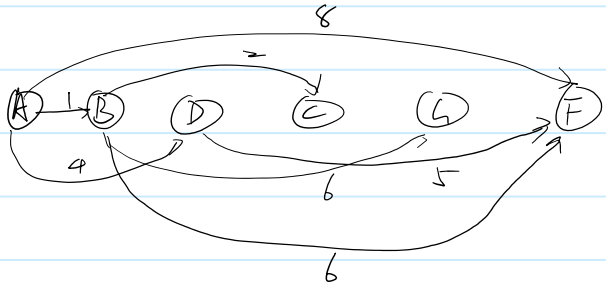
O(n+m)

8

2017年5月3日 15:44

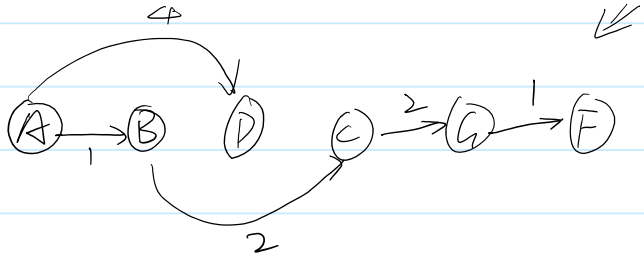
Ex:





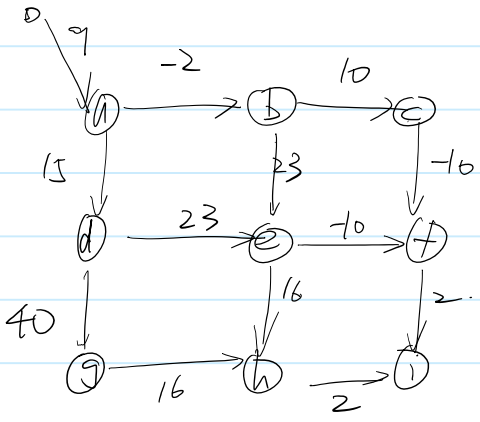
(D[v], Pred[v])	A	B	D	C	G	F
	(0, nil)	(+∞, nil)	(+∞, nil)	(+∞, nil)	(+∞, nil)	(+∞, nil)
		(1, A)	(4, A)			(8, A)
				(3, B)	(7, B)	(7, B)
					(5, C)	
						(6, G)

D-Value, Pred-value



Stair case problem.

$$\begin{bmatrix} 9 & -2 & 10 \\ 15 & 23 & -10 \\ 40 & 16 & 2 \end{bmatrix}$$



9

2017年5月3日 15:43

10

2017年5月3日 15:43

Week15

2017年5月8日 15:41

DAG-SP(G, w, s)

Compute a topological ordering of G

D[s] ← 0 , Pred[s] ← nil

O(n) { For each $v_i \neq s$,
  D[v_i] ← $+\infty$
  Pred[v_i] ← nil
endfor

O(n+m) { For i = 1 to n-1
  for each edge v_i, v_j .
    if D[v_j] > D[v_i] + w($v_i v_j$)
      D[v_j] ← D[v_i] + w($v_i v_j$)
      Pred[v_j] ← v_i
  endfor
endfor

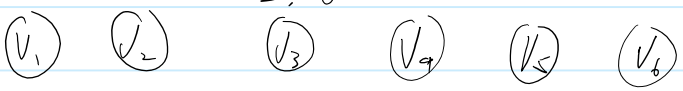
Return D-Value , Pred-Value

$O(|V| + |E|)$

$O(n+m)$

Works even the start point is in the middle of the topological ordering

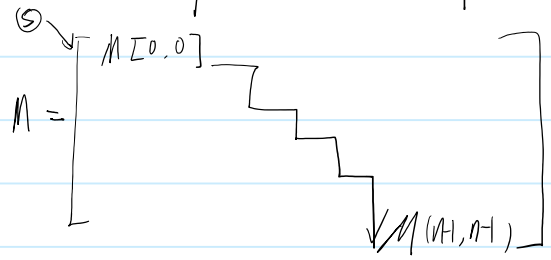
$+\infty$ $+\infty$ s, 0



Solve for longest path from s to M[n-1, n-1]

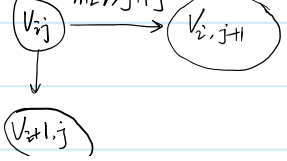
Staircase problem of M. Create a graph - G whose vertices are

$O(n^2+n^2) = O(n^2)$



v_{ij}, where v_{ij} "represent" M_{ij} for $0 \leq i, j \leq n-1$

- whose edges are $(v_{ij}, v_{i,j+1})$ for $0 \leq i \leq n-1$, $0 \leq j \leq n-2$.



$(v_{ij}, v_{i+1,j})$ for $0 \leq i \leq n-2$, $0 \leq j \leq n-1$

*Additionally, nodes to node No with weight M[0,0]

*Additionally, nodes to node N₀ with weight M[0,0]

M[i+1, j] ↓ $V_{i+1,j}$

$(V_{ij}, V_{i+1,j})$ for $0 \le i \le n-2$

$0 \le j \le n-1$

2017年5月8日 15:41

II. Directed graph with non-negative weights.

Dijkstra's AIG (G, W, s)

```
D[s] ← 0,   Pred[s] ← nil
for each node v≠s                    } O(n)
    D[v] ← +∞
    Pred[v] ← nil
Let R be a data structure
  Insert each node v as (D[v], Pred[v]) into R

While R is not empty
    Remove v s.t. D[v] has the smallest value among all the nodes in R
    for each edge vz s.t. z in R
        if D[z] > D[v] + w(vz)
            D[z] ← D[v] + w(vz)
            Pred[z] ← v
            update D[z] in R
    end for
end while
    Return D-value / Find-values.
```

$n^2 = m \log n$

$\frac{n^2}{\log n} = m$

Dense: if $m \gg \frac{n^2}{\log n}$, $m \log n \geq n^2$. ⇒ Better to use Linked-list

Sparse: if $m \ll \frac{n^2}{\log n}$, → use PQ

running time.

	R is a Linked List	R is a Priority Queue (Heap)
n insertions into R	$O(n)$	$O(n \log n)$ [$O(n)$]
n removeMins from R	$O(n^2)$	$O(n \log n)$
$O(m)$ key updates	$O(m)$	$O(m \log n)$
	$O(n^2+m)$ // $O(n^2)$	$O(m \log n)$ ← $O(m+n \log n)$

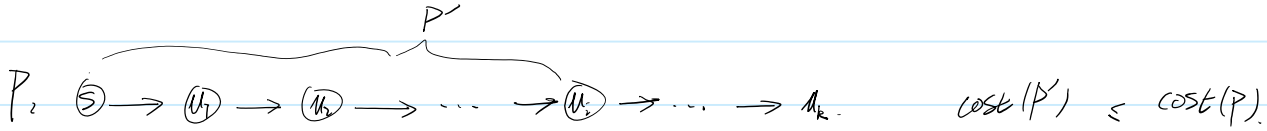
3

2017年5月8日 15:41

(not true actually)

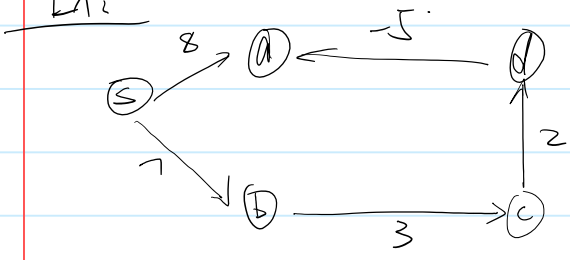
Ingredients:

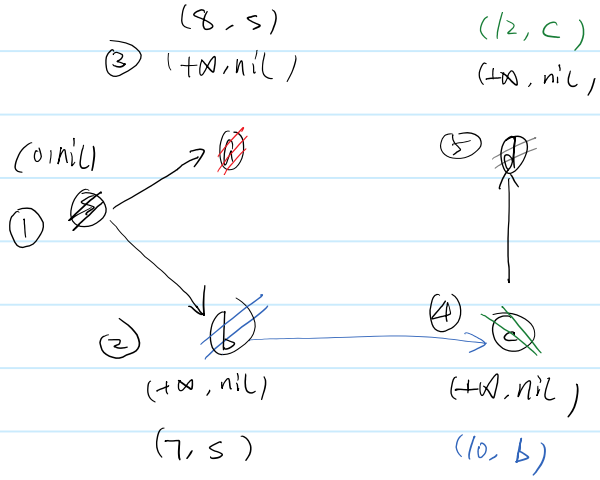
1. The past costs are monotonically (increasing) non-decreasing or monotonically (decreasing) non-increasing.



P is a sp from s to u_k

Ex:

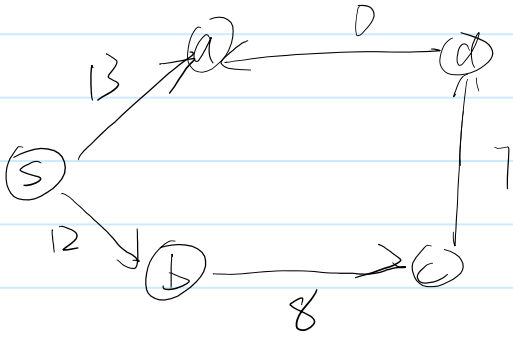




Find sp distances from s.

solve sp problem.
not Longest path problem

+5 to each edges



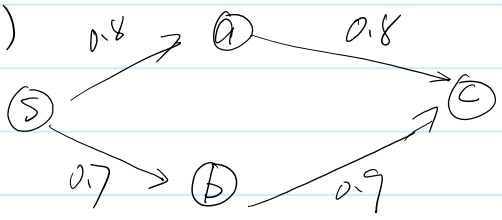
2. When paths are mon. non-decreasing — minimization problem
mon non-increasing — maximization problem.

4

2017年5月8日 15:41

EX # 1

$G = (V, E)$



$P = e_1, e_2, \dots, e_k$.

$rel(P) = P(e_1) \times P(e_2) \times \dots P(e_k)$

Most reliable path.

$S \to a \to C \quad 0.8 \times 0.8 = 0.64$ ✗

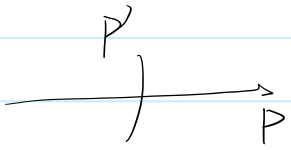
$S \to b \to C \quad 0.7 \times 0.9 = 0.63$.

Check ingredients:

#1. Path cost monotonicity

Let $P = e_1, e_2, \dots e_k$.

$P' = e_1, e_2, \dots e_i$.



$rel(P') \geq rel(P)$

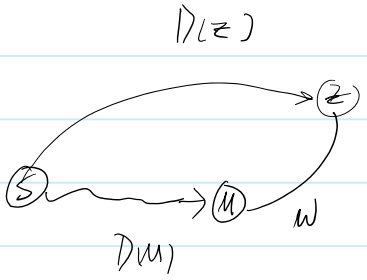
$\Rightarrow$ Path costs are non non-increasing..

Let $D[u]$ represent the reliability of the best path from s to u.

5

2017年5月8日 15:41

```
Reliability (G, w, s)
  D[s] ← 1 , Pred[s] ← nil
  For each node v ≠ s
      D[v] ← 0
  end for  Pred[v] ← nil
  Insert all nodes into maxpQ Q
  While Q is not empty
      u ← removeMax(Q)
    for each edge uz , z in Q
    if D[z] < D[u] × w(uz)
      D[z] ← D[u] × w(uz)
      Pred[z] ← u
      update D[z)
  end while.
```

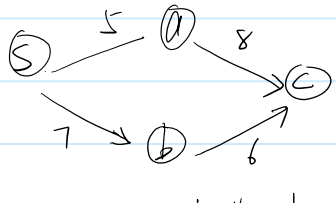


6

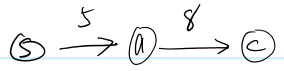
2017年5月10日 15:41

EX #2 Max capacity paths

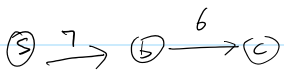
Given a directed graph G. and for each edge e,
it has an capacity $c(e) > 0$



bottleneck



5



6

"power wire can carry"
"bandwidth of link"
"weight capacity"
"# of cars"

$P = e_1, e_2, e_3, \cdots e_k$

$Cap(P) = \min_{1 \le i \le k} \{c(e_i)\}$

↳ bottleneck in P

Pick paths to every node. So that their capacities are as large as possible.

Check Ingredients:

#1. Monotonicity of path cost.

$P: e_1, e_2, \cdots e_i - e_k$.

$P': e_1, e_2 - e_i$.

$Cap(P) \le Cap(P')$

* montonically non-increasing (decreasing)

↓↓

Maximization Problem.

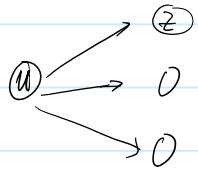
7

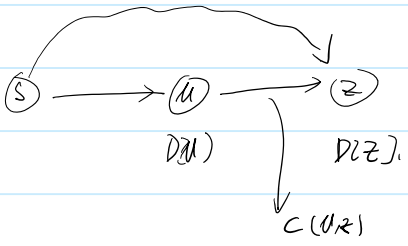
2017年5月10日 15:41

Maximization problem.

Let D[u] represent the capacity of the best path found so far by the algorithm.

```
D[s] ← +∞
Pred[s] ← nil
for each node v ≠ s
    D[v] ← 0
    Pred[v] ← nil
Store all nodes in a max PQ with D-values as keys.
While Q is not empty.
    u ← removeMax(Q)
    for each edge uz st. z ∈ Q
        if D[z] < min{D[u], c(uz)}
            D[z] ← min{D[u], c(uz)}
            Pred[z] ← u.
            update D[z] in Q
    end for
end while
```





8

2017年5月10日 14:27

HW # 11.

① n days, b bids. — i^{th} bid. (s_i, t_i, m_i)
start end bid.

sol'n #1:

Vertices: 1, 2 — n.

edges: for $i = 1$ to b. connect s_i to t_i.

for $j = 1$ to $n-1$, connect j to $j+1$ with weight 0

Solve the longest path in this DAG

$|V| = n$. $|E| = b + (n-1)$

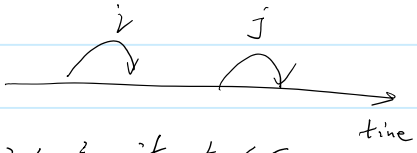
$O(n + b + n-1)) = O(n+b)$

build the graph

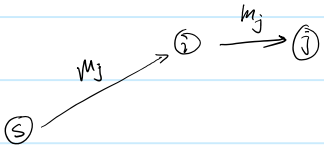
$O(n + b + (n-1)) = O(n+b)$

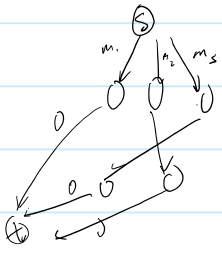
Soln #2. S

Vertices: 1, 2, --- b. represent bids.



Edges: For every pair $\{i, j\}$, connect i to j if $t_i \in S_j$
and make cost equal to m_j.





For each bid i, connect s to i with cost m_i.

construct graph. $O(b + b^2) = O(b^2)$.
node edge

solve longest path problem.

$O(b + b^2) = O(b^2)$

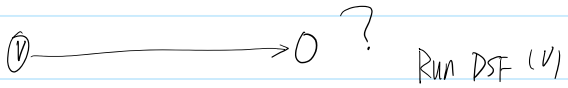
Overall, $O(b^2)$

9

2017年5月10日 14:27

4. T/F

(c) False (d) True



6. Best shortest paths best[u] = min # of edges in a sp problem from s to u.

D[s] ← 0, pred[s] ← nil, best[s] ← 0

For each node v ≠ s

D[v] ← +∞

Pred[v] ← nil.

best[v] ← +∞ 2 Dimension.

Insert all nodes into a min-PQ with (D(v), best[v]) as keys

while Q is not empty

v ← removeMin(Q)

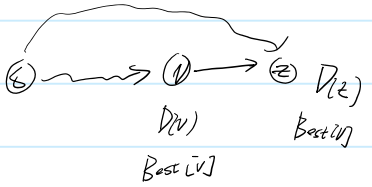
for each z s.t. vz is an edge and z ∈ Q

if D[z] > D[v] + w(vz)

D[z] ← D[v] + w(vz)

pred[z] ← v

best[z] ← best[v] + 1



else

if D[z] = D[v] + w(v,z) and best[v] + 1 < best[z]

best[z] ← best[v] + 1

pred[z] ← v

update d[z]

endfor

end while

10

2017年5月10日 14:27

2017年3月1日 15:58

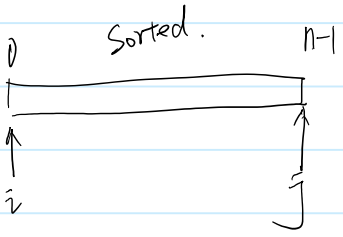
Q4.

$S[i] + S[j] = z$

(1) Brute Force , $C(n,2) = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$

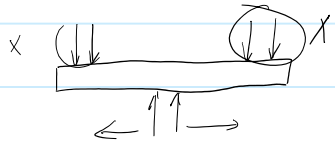
(2)
For $i \leftarrow 0$ to $n-2$
  $x \leftarrow z - S[i]$
  Search for x in $S[i+1 \cdots n-1]$ using Binary Search.
  If x exsits
    Return "yes"
End for
Return "no"

$O(n \log n)$




```
i ← 0
j ← n-1
while i < j
    if S[i] + S[j] > z.
        j ← j-1
    else
        if S[i] + S[j] < z.
            i ← i+1
        else
            return "yes"
end while.
Return "no"
```

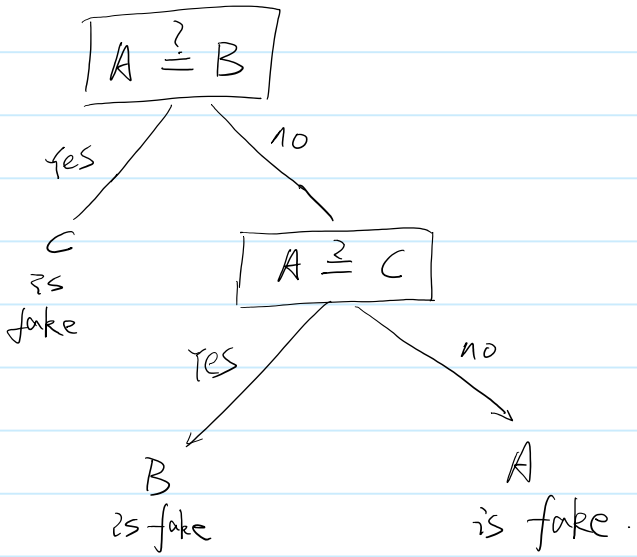
```
i ← ⌊n/2⌋
j ← ⌊n/2⌋ + 1
while i ≥ 0 & j ≤ n-1.
    if S[i] + S[j] > z.
        i ← i-1
    else
        if S[i] + S[j] < z.
            j ← j+1.
        else
            Return "yes"
end while.
Return "no"
```



2017年1月23日 15:57

HW #1 problem 3.

a. $A\{C_1\ C_2\ C_3\}$
$B\{C_4\ C_5\ C_6\}$
$C\{C_7\ C_8\ C_9\}$



4 weighing

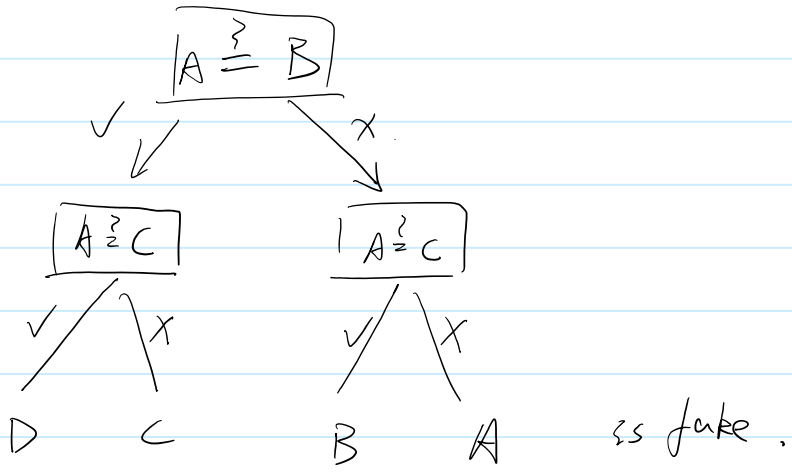
b. $A=\{C_1, C_2 \cdots C_{\frac{n}{3}}\}$

$B=\{C_{\frac{n}{3}+1} \text{ —— } C_{\frac{n}{3}\times 2}\}$

$C=\{C_{\frac{2n}{3}+1} - - C_{3\cdot\frac{n}{3}}\}$

$D=\{C_{n-1} \cdots C_n\}$

$|D| \le 2$



If $n \ge 6$,

$n \to \frac{n}{3} \to \frac{n}{9} \to \frac{n}{3^3} - - \frac{n}{3^k}$

to find the smallest k

$\frac{n}{3^k} \le 1$

$n \le 3^k$

$\log_3 n \le \log_3 3^k$

$\log_3 n \le k$

$\therefore \quad 2 \times \lceil \log_3 n \rceil$

1. $n^{\frac{2}{3}}$, 1.5^n, n^{100}, $(\log n)^3$, $\sqrt{n}\log n$, $n^{99}+n^{98}$, $9^{\log_3 n}$, $n!$

close Notation $O(1)$, $O(2^n)$, $O(100^n)$, $O(n^{99})$, $O(n^2)$, $\Omega(a^n)$, $\Omega(1.5^n)$

Loose Notation: $O(n)$ $O(n^{100})$ $O(n^3)$ $O(n^{\frac{3}{2}})$ $O(n^{99})$ (Polynomial)

$9^{\log_3 n} = n^{\log_3 9} = n^2$

$$\lim_{n\to\infty}\frac{(\log n)^3}{\sqrt{n}\log(n)} = \lim_{n\to\infty}\frac{\log n^2}{\sqrt{n}} = \lim_{n\to\infty}\frac{2\log n\cdot\frac{1}{n\ln 2}}{\frac{1}{2}n^{-\frac{1}{2}}} = \lim_{n\to\infty} = \frac{4}{\ln 2}\cdot\frac{\log n}{\sqrt{n}} = \lim_{n\to\infty}\frac{4}{\ln 2}\cdot\frac{1}{n\ln 2}\cdot 2\sqrt{n} = \lim_{n\to\infty}\frac{8}{(\ln 2)^2\ln n} = 0$$

∴ growth rate of $\sqrt{n}\log n$ is greater than $(\log n)^3$.

$$\lim_{n\to\infty} = \frac{\sqrt{n}\log n}{n^{\frac{2}{3}}} = \lim_{n\to\infty}\frac{\log n}{n^{\frac{1}{6}}} = \lim_{n\to\infty}\frac{\frac{1}{n\ln 2}}{\frac{1}{6}n^{-\frac{5}{6}}} = \lim_{n\to\infty} = \frac{6\cdot n^{\frac{5}{6}}}{\ln 2\cdot n} = \lim_{n\to\infty}\frac{6}{\ln 2}\cdot n^{-\frac{1}{6}} = 0$$

a. $(\log n)^3$, $\sqrt{n}\log n$, $n^{\frac{2}{3}}$, $9^{\log_3 n}$, $n^{99}+n^{98}$, n^{100}, 1.5^n, $n!$ (based on close notation and limit value)

b. polynomial: $n^{\frac{2}{3}}$, $9^{\log_3 n}$, $n^{99}+n^{98}$, n^{100}, (based on loose notation)

c. expotential: 1.5^n, $n!$

2.

a. # of iteration is $\lceil\log_{b^2} n\rceil$. smaller than $\lceil\log_b n\rceil$, when $b>1$.

$$\lim_{n\to\infty}\frac{\log_{b^2} n}{\log_b n} = \frac{\log_c n}{\log_c b^2}\times\frac{\log_c b}{\log_c n} = \log_{b^2} b = \frac{1}{2}$$

∴ $\log_{b^2} n$ is $\theta(\log_b n)$ and $\lceil\log_{b^2} n\rceil$ is $\theta(\log_b n)$

b. # of iteration is $\lceil\log_{\frac{1}{b}} n\rceil$ greater than $\lceil\log_b n\rceil$, when $b>1$

$$\lim_{n\to\infty}\frac{\log_{\frac{1}{b}} n}{\log_b n} = \frac{\log_c n}{\log_{\frac{1}{b}}}\times\frac{\log_c b}{\log_c n} = \log_{\frac{1}{b}} b \text{ is constant when } b>1$$

∴ $\log_{\frac{1}{b}}$ is $\theta(\log_b n)$ and $\lceil\log_{\frac{1}{b}}\rceil$ is $\theta(\log_b n)$

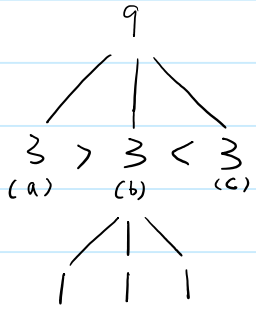
c. # of iteration is $\lceil n/b\rceil$, It is $\Omega(\log_b n)$

$\frac{n}{b} \geq c\cdot\log_b n$ (when $b>1$, $c=1$, $n_0>1$)

$n \geq b\cdot c\cdot\log_b n$ (when $c>0$, $b\cdot c=1$, $n_0>1$) ∴ n/b is $\Omega(\log_b n)$.

3. a. when n=9. , the fewest weighing in worst case is 3

Worst case.



divide 9 coin evenly into 3 group, named a, b, c.

1st weighting: compare a and b, if their weights are not equal, then write down which group is lighter (assume fake coin is lighter, vice versa).

2nd weighting: compare b and c, if their weights are equal, then a contains fake coin, if b is lighter than c, then group b contains fake coin.

3nd weighting: divide b into 1, 1, 1 group, randomly pick two, if not equal, the lighter one is fake. if equal, then the unchosen coin is fake.

b. In order to generize the method, we only consider the weight of coins, that can solve times of weighting in the following steps. And we don't consider the situation of coin combination.

Basically, just divide n coins evenly into 3 parts, which will 3 situation, 0, 1, 2 remainders. And branching to the smallest integer,

1st step

n/3 remainder	0	1	2
times of weighting	2	2	2

2~($\log_2 n - 1$) step

remainder	0	1	2
times of weighting	1	2	2

last step

times	remainder 0	1	2
1	1	2	2
2	3	3	3

when n ≥ 9, steps, $\log_3 n$

when $\log_3 n \geq 3$ ① smallest times: $2 + (\log_3 n - 1) \times 1 + 1 = \log_3 n + 2$

② biggest times: $2 + (\log_3 n - 1) \times 2 + 3$

$= 3 + 2\log_3 n$

∴ the weighting times is $O(\log_3 n)$

4. $f(n)$ is $O(g(n))$, $f(n) \leq c \cdot g(n)$, when $n \geq n_0$ ①

$d(n)$ is $O(e(n))$, $d(n) \leq c \cdot e(n)$, when $n \geq n_0$ ②

a.

①+② $f(n) + d(n) \leq c \cdot g(n) + c e(n)$

$f(n) + d(n) \leq c \cdot (g(n) + e(n))$ when $n \geq n_0$, real constant $c > 0$

∴ $f(n) + d(n)$ is $O(g(n) + e(n))$

∵ $f(n) > 0$, $d(n) > 0$

b. ①×②

$f(n) \cdot d(n) \leq c^2 g(n) \cdot e(n)$ when $n \geq n_0$, real constant $c^2 > 0$.

∴ $f(n) \cdot d(n)$ is $O(g(n) \cdot e(n))$